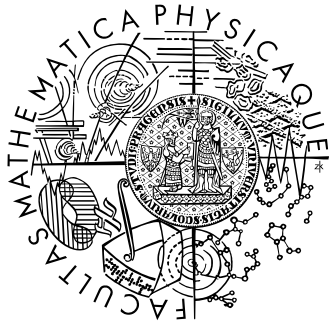
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Bc. Miroslav Jakubík

RBF-sítě s dynamickou architekturou

Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: doc. RNDr. Iveta Mrázová, CSc.

Studijní program: Informatika, Teoretická informatika

2012

Chcel by som poďakovať všetkým, ktorí mi pomohli s touto prácou. V prvom rade vedúcej mojej práce doc. RNDr. Ivete Mrázovej, CSc. za trpezlivosť a množstvo dobré mierených rád. Ďalej by som chcel poďakovať za podporu celej mojej rodine a priateľom.

Prehlasujem, že som svoju diplomovú prácu napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím so zapožičiavaním práce.

V Prahe dňa Bc. Miroslav Jakubík

Obsah

Názov práce: RBF-sítě s dynamickou architekturou
Autor: Bc. Miroslav Jakubík
Katedra: Katedra teoretické informatiky a matematické logiky
Vedúca diplomovej práce: doc. RNDr. Iveta Mrázová, CSc.
e-mail vedúcej: Iveta.Mrazova@mff.cuni.cz

Abstrakt: V tejto diplomovej práci som zrekapituloval viacero metód vhodných pre klastrovanie dát. Predstavil som dva dobré známe klastrovacie algoritmy, a to konkrétne K-means algoritmus a Fuzzy C-means (FCM) algoritmus. Uviedol som niekoľko metód vhodných pre odhad optimálneho počtu klastrov. Ďalej som predstavil základný model Kohonenových máp a dva modely Kohonenových máp s adaptívnou topológiou, konkrétne Kohonenové mapy s rastúcou mriežkou a model rastúcich neurónových plynov. Ako posledný som predstavil relatívne nový model radiálne bázických neurónových sieti. Pre tento typ neurónových sieti som uviedol viaceré učiace algoritmy RAN, RANKEF, MRAN, EMRAN a GAP. V závere práce som aplikoval jednotlivé klastrovacie metódy na reálne dáta popisujúce vzájomný obchod štátov sveta.

Kľúčové slová: klastrovanie, K-means algoritmus, Fuzzy C-means klastrovanie, Kohonenové siete, radiálne bázické neurónové siete, GAP algoritmus

Title: RBF-networks with a dynamic architecture
Author: Bc. Miroslav Jakubík
Department: Department of Theoretical Computer Science and Mathematical Logic
Supervisor: doc. RNDr. Iveta Mrázová, CSc.
Supervisor's e-mail address: Iveta.Mrazova@mff.cuni.cz

Abstract: In this master thesis I recapitulated several methods for data clustering. Two well known clustering algorithms, concretely K-means algorithm and Fuzzy C-means (FCM) algorithm, were described in the submitted work. I presented several methods, which could help estimate the optimal number of clusters. Further, I described Kohonen maps and two models of Kohonen's maps with dynamically changing structure, namely Kohonen map with growing grid and the model of growing neural gas. At last I described quite new model of radial basis function neural networks. I presented several learning algorithms for this model of neural networks, RAN, RANKEF, MRAN, EMRAN and GAP. In the end of this work I made some clustering experiments with real data. This data describes the international trade among states of the whole world.

Keywords: clustering, K-means algorithm, Fuzzy C-means clustering, Kohonen networks, radial basis neural networks, GAP algorithm

1. Úvod

Analýza a spracovanie dát sa stala bežnou súčasťou mnohých procesov v rôznych odvetviach. Firmy, organizácie, výskumné ústavy uchovávajú množstvo dát vo svojich databázach. Zvyčajne však tieto dáta nevznikajú s cieľom ich analyzovať, ale sú výsledkom rôznych procesov ako napríklad obchodných, marketingových alebo výrobných. Často sa stáva, že tieto dáta môžu byť poškodené, časť informácií chýba, prípadne niektoré informácie sa duplikujú. K analýze dát, objavovaniu určitých vnútorných štruktúr a vzájomných závislostí sa využívajú aj klastrovacie techniky.

Z oblasti, ktoré sa dotýkajú takmer každého z nás, sa klastrovanie dát využíva napríklad v poisťovníctve, bankovníctve, marketingu ale aj zdravotníctve. Na základe získaných výsledkov je možné vytvoriť a ponúknuť atraktívnejšie produkty, služby presne cieleným skupinám potenciálnych klientov.

V tejto diplomovej práci si predstavíme viacero klastrovacích metód. Tieto metódy budeme analyzovať a vzájomne porovnávať podľa viacerých kritérií ako rýchlosť, zložitosť a výsledná presnosť. Zameriame sa na vhodnosť použitia jednotlivých metód na rôznych typoch dát, aké informácie potrebujú poznať, aby mohli byť použité. Diplomová práca je rozčlenená do nasledujúcich kapitol.

V druhej kapitole nájdeme stručný úvod do problematiky klastrovania, základné delenie klastrovacích techník. Uvedieme mechanizmus, ktorý nám umožní pracovať s dátami.

V tretej kapitole predstavíme K-means algoritmus (0) a metódy, ktoré môžu pomôcť pri hľadaní optimálneho počtu klastrov.

Vo štvrtej kapitole predstavíme FCM algoritmus (4). Uvedieme niekoľko kritérií validity klastrov, ktoré hodnotia kvalitu rozdelenia vstupných dát do klastrov.

V piatej kapitole predstavíme Kohonenové mapy (5), ktoré sú jedným z hlavných predstaviteľov neurónových sieti typu SOM (Self-Organizing Maps).

V šiestej kapitole predstavíme dve varianty Kohonenových máp s adaptívnou topológiou. Konkrétne Kohonenové mapy s rastúcou mriežkou (6.1) a model rastúcich neurónových plynov (6.2).

V siedmej kapitole predstavíme jeden z mladších modelov neurónových sieti, a to RBF siete (Radial Basis Function). Uvedieme dva algoritmy učenia RBF neurónových sieti náhodná voľba stredov RBF jednotiek (7.4.1) a hybridné učenie (7.5).

V ôsmej kapitole predstavíme RBF siete s adaptívnou topológiou. Uvedieme viacero učiacich algoritmov RAN (8.1), RANEEKF (8.1), MRAN (8.2), EMRAN (8.3), GAP (8.4), RGAP (8.4.1). Ďalej si predstavíme metódy, ktoré možno využiť pri samotnom

návrhu architektúry RBF sieti. Uvedieme citlivostnú analýzu (8.7) a algoritmus kontextového klastrovania (8.9), ktorý pomáha riešiť problém mnohorozmernosti vstupných dát.

V deviatej kapitole prevedieme testovanie niektorých z uvedených klastrovacích metód. Testy prevedieme najskôr na testovacej množine umelo vygenerovaných trénovacích vzorov. Následne otestujeme klastrovacie metódy na reálnych dátach, ktoré popisujú vzájomné obchodovanie jednotlivých štátov sveta.

V desiatej kapitole predstavíme program Klastrovacie techniky, ktorý umožňuje testovanie uvedených klastrovacích metód. Program Klastrovacie techniky je súčasťou tejto diplomovej práce.

2. Klastrovanie

Pod pojmom klastrovanie rozumejme proces zaraďovania konkrétnych, či abstraktných objektov do navzájom disjunktných skupín (nazývaných aj ako klastre) na základe pozorovania ich vlastnosti. Získané klastre nám pomôžu objaviť vnútornú štruktúru medzi spracovávanými objektmi.

Klastrom budeme nazývať skupinu objektov, ktoré sú si navzájom čo najviac podobné a zároveň sa tieto objekty čo najviac odlišujú od objektov umiestnených vo zvyšných klastroch.

Po klastrovacích algoritmoch obvykle požadujeme, aby výsledné rozdelenie objektov do klastrov bolo čo najviac jednoznačné. Teda, aby získane klastre boli pokiaľ je to možné jasne ohraničené a vzájomne sa čo najmenej prekrývali. Existujú však aj algoritmy, ktoré umožňujú aby jeden objekt patril súčasne do viacerých klastrov. Príkladom je FCM algoritmus (4).

Zaradenie objektov do jednotlivých klastrov sa riadi podľa istých kritérií, ktoré nám určujú kvalitu tohto rozdelenia. Keďže v oblasti dobývania znalostí pracujeme so vstupnými dátami, ktoré sú reprezentované pomocou vektorov, naším kritériom podobnosti bude vzdialenosť dvoch vektorov v priestore. Čím je vzájomná vzdialenosť vektorov menšia, tým sú si viac podobné, a naopak, čím je ich vzájomná vzdialenosť väčšia, tým viac sa od seba odlišujú. Vzdialenosť dvoch vektorov môžeme počítať pomocou niektorej z nižšie uvedených metrík, pričom ich použitie je závislé od konkrétneho typu zadaného problému.

Nech $\vec{p}, \vec{q}$ sú dva n-rozmerné vstupné vektory $\vec{p} = [p_1, p_2, \dots, p_n]$, $\vec{q} = [q_1, q_2, \dots, q_n]$. Vzdialenosť týchto dvoch vektorov môžeme spočítať v jednotlivých metrikách pomocou nasledujúcich vzorcov:

- Euklidova vzdialenosť

$$d_E(\vec{p}, \vec{q}\,) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{2.1}$$

- Manhattanska vzdialenosť

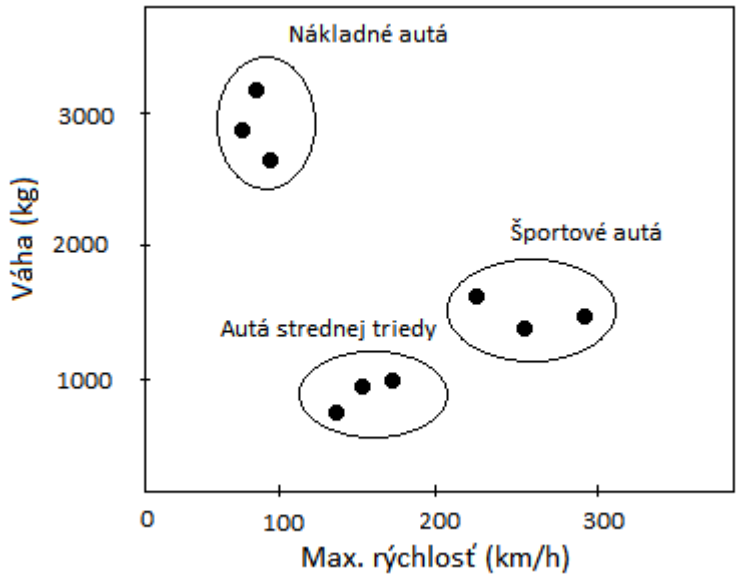
$$d_M(\vec{p}, \vec{q}\,) = \sum_{i=1}^{n} |p_i - q_i| \tag{2.2}$$

- Kosínova vzdialenosť

$$d_S(\vec{p}, \vec{q}\,) = \frac{\sum_{i=1}^{n} p_i \cdot q_i}{\sqrt{\sum_{i=1}^{n} {p_i}^2 \cdot \sum_{i=1}^{n} {q_i}^2}} \tag{2.3}$$

- Čebyševova vzdialenosť

$$d_C(\vec{p}, \vec{q}) = max_{i=1,\dots,n} |p_i - q_i| \tag{2.4}$$



Obrázok č. 1: Jednoduchý príklad klastrovania. Jednotlivé klastre tvoria vozidlá rôznych kategórií.

3. K-means clustering (K-priemerová metóda)

K-priemerová metóda [33] patrí medzi jeden z najznámejších a často využívaných klastrovacích algoritmov. Zaraďuje sa k metódam typu učenia bez učiteľa.

3.1. Chybová funkcia pre K-means algoritmus

Nech $X = \{\vec{x}_p \in R^I, p = 1, \dots, P\}$ je množina P vstupných vektorov rozdelených do K zhlukov v_k, $k = 1, \dots, K$. Predpokladá sa $K \ll P$. Každý zhluk v_k je reprezentovaný vektorom $\vec{c}_k$, ktorý je umiestnený do ťažiska zhluku v_k. Vektory $\vec{c}_k, k = 1, \dots, K$ sa nazývajú centroidy príslušných zhlukov.

Chybová funkcia E_{KM} K-means algoritmu vyjadruje súčet vzdialenosti medzi jednotlivými vstupnými vektormi $\vec{x}$ a im odpovedajúcimi centroidami $\vec{c}_k, k = 1, \dots, K$. Funcia E_{KM} je daná predpisom:

$$E_{KM} = \sum_{k=1}^{K} \sum_{\vec{x} \in v_k} \|\vec{x} - \vec{c}_k\|^2 \tag{3.1}$$

$\|\vec{x} - \vec{c}_k\|$ určuje vzdialenosť medzi vstupným vektorom $\vec{x}$ a centroidom $\vec{c}_k$ v zvolenej metrike.

Cieľom K-means algoritmu je minimalizovať hodnotu chybovej funkcie E_{KM}. Hodnota tejto funkcie typicky klesá s postupným pridávaním ďalších zhlukov. Tento pokles však po prekročení optimálneho počtu zhlukov začína byť len nepatrný.

3.2. Algoritmus K-means

Na začiatku algoritmu [32], [46] užívateľ určí požadovaný počet zhlukov K, do ktorých sa rozdelia vstupné vektory $\vec{x}$. Vo vstupnom priestore náhodne vygenerujeme K vektorov a tie prehlásime za centroidy $\vec{c}_k, k = 1, \dots, K$. Pre každý vstupný vektor $\vec{x}_p \in X$ nájdeme centroid $\vec{c}_k$, ktorý je najbližšie k tomuto vektoru podľa vzťahu:

$$c_k = \arg\min_k \left\| \vec{x}_p - \vec{c}_k \right\|^2, \quad k \in \{1, .., K\} \tag{3.2}$$

Následne tento vstupný vektor $\vec{x}_p$ zaradíme do zhluku v_k reprezentovaného centroidom $\vec{c}_k$. Po zaradení všetkých vstupných vektorov pristúpime k prepočítaniu pozícií všetkých centroidov, a to v zmysle, že centroid $\vec{c}_k$ posunieme do ťažiska príslušného zhluku v_k. Tento postup opakujeme dovtedy, pokiaľ sa menia pozície centroidov.

Pseudo – kód: K-means algoritmus

1. Inicializácia:
   Náhodne vygeneruj K navzájom rôznych centroidov $\vec{c}_k, k = 1, \dots, K$..

2. Priradenie do zhluku:
   Pre každý vstupný vektor $\vec{x}_p \in X$ nájdi najbližší centroid $\vec{c}_k$ a priraď vektor $\vec{x}_p$ do príslušného zhluku v_k. Index najbližšieho centroidu $\vec{c}_k$ k vektoru $\vec{x}_p$ spočítaj podľa predpisu:

$$c_k = \arg\min_k \left\| \vec{x}_p - \vec{c}_k \right\|^2, \quad k \in \{1, .., K\} \tag{3.3}$$

3. Aktualizácia pozícií centroidov:
   Vypočítaj nové pozície centroidov podľa predpisu:

$$\vec{c}_k = \frac{1}{n_k} \sum_{\vec{x} \in v_k} \vec{x}, \quad k = 1, \dots, K \tag{3.4}$$

   n_k určuje počet vstupných vektorov, ktoré boli v tejto iterácii algoritmu priradené do zhluku v_k reprezentovaného centroidom $\vec{c}_k$.

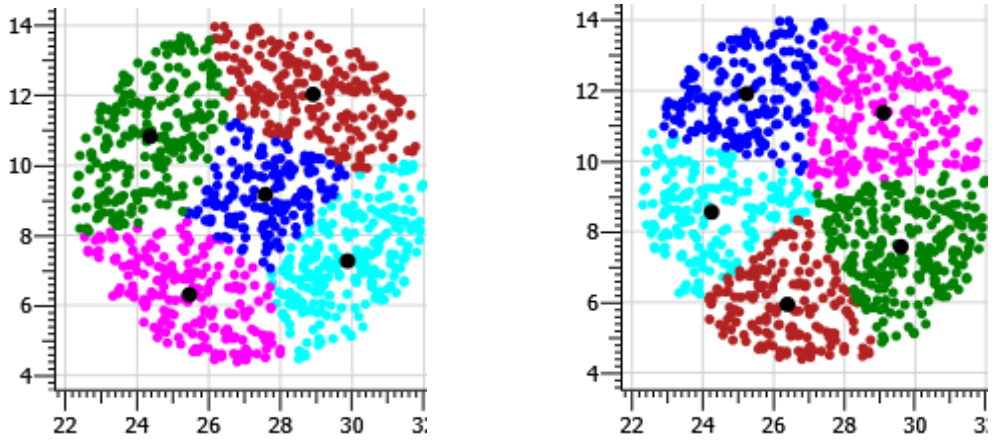
4. Ukončenie algoritmu:
   Pokiaľ došlo k zmene jednotlivých pozícií centroidov, pokračuj na krok algoritmu číslo 2. Ak sa pozície centroidov ustálili, ukonči algoritmus.

Inicializačný krok tohto algoritmu je možné nasledovne pozmeniť. Generovanie náhodných umiestnení centroidov jednotlivých zhlukov úplne vypustíme a za centroidy $\vec{c}_k$ náhodne zvolíme niektoré vektory z množiny vstupných vektorov X.

3.3. Vlastnosti K-means algoritmu

K-means algoritmus jednoznačne určí príslušnosť každého vstupného vektoru ku konkrétnemu zhluku, nie je teda možné, aby jeden vektor zároveň patril do viacerých zhlukov.

Medzi hlavné výhody patrí jednoduchá implementácia algoritmu a rýchlosť výpočtu. Vďaka týmto vlastnostiam je možné K-means algoritmus opakovane použiť aj na spracovanie dát väčšieho objemu. Opakované spustenie algoritmu na rovnakých dátach nám však nezaručuje, že dospejeme k rovnakým výsledkom. Obrázok č. 2 ilustruje rôzne výsledky K-means algoritmu na rovnakých vstupných dátach.



Obrázok č. 2: Opakované spustenie K-means algoritmu na rovnakých dátach môže viesť k rozličnému rozloženiu klastrov vo vstupnom priestore. Čierne body predstavujú umiestnenie jednotlivých centroidov príslušných klastrov.

Nevýhodou K-means algoritmu je, že často nenájde optimálne riešenie, pretože sa zastaví v nejakom lokálnom optime (avšak často už po niekoľkých iteráciách). Toto môžeme čiastočne odstrániť viacnásobným spustením algoritmu na rovnakých dátach, avšak s iným náhodným rozložením centroidov.

Výkonnosť tohto algoritmu veľmi závisí na inicializačnej fáze. Keďže užívateľ musí dopredu určiť počet klastrov, je potrebné odhadnúť optimálny počet zhlukov a optimálne rozmiestnenie ich centroidov vo vstupnom priestore. Pri zvolení menšieho počtu zhlukov niektoré zhluky nebudú objavené, ale budú zahrnuté v iných objavených zhlukoch. Pri voľbe väčšieho počtu zhlukov by sa pri optimálnom rozložení centroidov mali objaviť všetky dostupné zhluky plus naviac nejaké zhluky obsahujúce len centroidy.

Optimálne rozmiestnenie centroidov

Veľmi dôležitou otázkou teda je ako určiť optimálne rozmiestnenie centroidov vo vstupnom priestore. Jednou z možností je experimentovanie, čiže viacnásobné spustenie algoritmu vždy s iným náhodným rozmiestnením centroidov a po porovnaní dosiahnutých výsledkov vyberieme najlepší variant. Ďalšou možnosťou je postupné rozmiestňovanie centroidov vo vstupnom priestore. Prvý centroid umiestnime náhodne do vstupného priestoru, druhý centroid umiestnime na takú pozíciu, ktorá je čo najviac vzdialená od prvého centroidu. Podobne postupujeme s umiestňovaním ďalších centroidov, teda j-tý centroid umiestnime tak, aby bol čo najvzdialenejší od $j-1$ predchádzajúcich centroidov.

Optimálny počet zhlukov

Nie je známe obecné pravidlo, ktoré by nám jednoznačne určilo optimálny počet zhlukov. Veľmi jednoduchým pravidlom [56] je pravidlo hrubého odhadu, vyjadrené predpisom:

$$K \approx \sqrt{\frac{P}{2}}, P \text{ je počet vstupných vektorov} \tag{3.5}$$

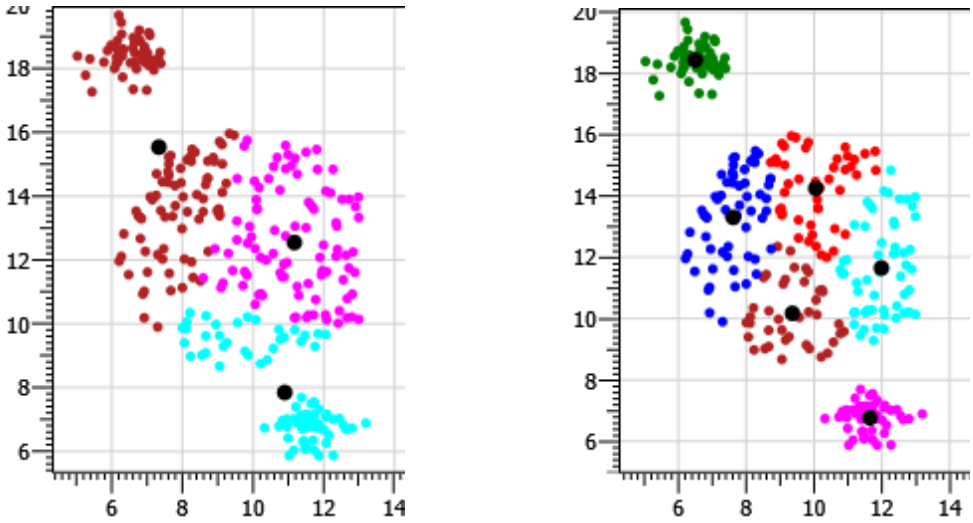
Ďalšou možnosťou je uhlové kritérium, v anglickej literatúre označované ako Elbow criterion. Táto metóda funguje v princípe nasledovne. Postupným zväčšovaním počtu klastrov sa zväčšuje množstvo získaných informácií o vstupných dátach a zároveň sa týmto zmenšuje hodnota chybovej funkcie E_{KM}. Táto zmena chybovej funkcie je pomerne značná, kým sa nedosiahne optimálny počet klastrov. S ďalším nárastom klastrov však táto zmena nastáva stále pomalšie. Cieľom je určiť taký počet klastrov, že pridanie ďalšieho klastra už neprinesie výrazne väčšie množstvo informácií o vstupných dátach. Tento počet klastrov možno určiť z grafu, aj keď nie vždy jednoznačne. Vytvoríme si graf závislosti chybovej funkcie (3.1) na počte klastrov a hľadáme taký bod grafu, v ktorom je najväčší zlom na krivke (najväčšia uhlová zmena).

Graf č. 1 Uhlové kritérium, najväčší zlom na krivke označuje optimálny počet klastrov.

Problémové vstupné dáta

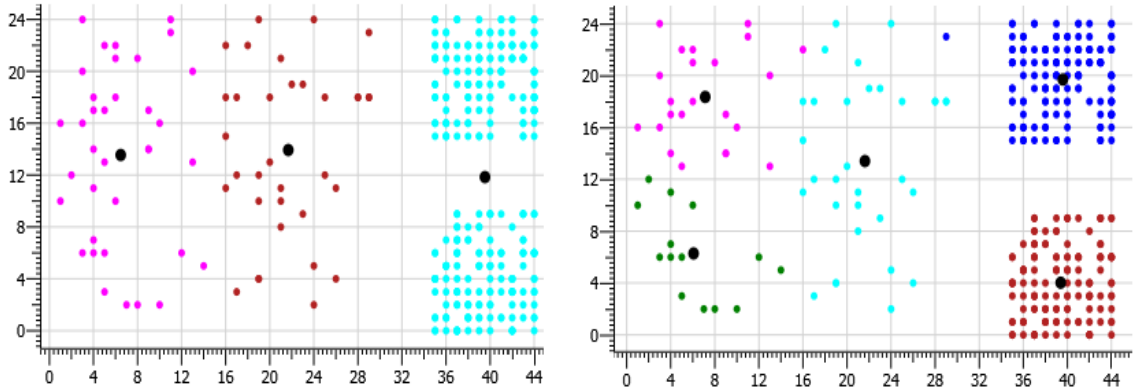
K-means algoritmus vytvára kompaktné klastre guľového tvaru. Bohužiaľ však existujú vstupné dáta, pre ktoré tento algoritmus nenájde správne rozloženie zhlukov. Možným riešením tohto problému môže byť zväčšenie počtu klastrov a ich následne zlučovanie, prípadne použiť inú vhodnejšiu klastrovaciu metódu. Nasledujúce obrázky 5 - 7 znázorňujú niektoré prípady vstupných dát, na ktorých použitie K-means algoritmu nemusí viesť k správnemu rozdeleniu vstupných dát do klastrov.

Problémove môžu byť dáta, ktoré vytvárajú klastre rozličnej veľkosti. V takomto prípade môže dôjsť k nesprávnemu zaradeniu vstupných vektorov z klastrov väčších rozmerov do klastrov menších rozmerov. Obrázok č. 3 zachytáva práve takýto prípad.



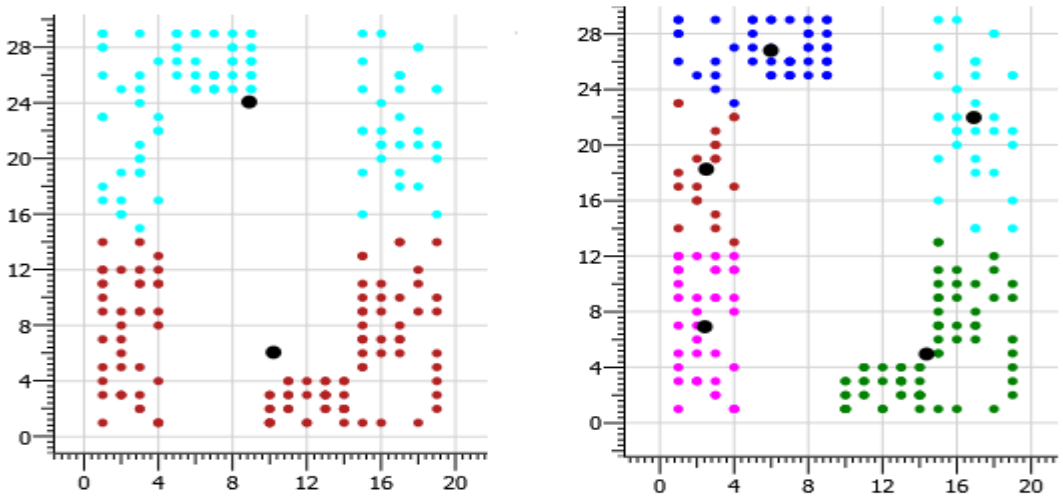
Obrázok č. 3. Problémy K-means algoritmu s klastrami rôznej veľkosti. Vľavo vidíme nesprávne rozdelenie vstupných dát do 3 klastrov. Vpravo vidíme rozdelenie vstupných dát do 6 klastrov, pričom klastre menšej veľkosti sú už určené správne.

Problémy s rozdelením vstupných dát do klastrov sa môžu vyskytnúť aj pri klastroch rôznej hustoty. Obrázok č. 4 zachytáva tento prípad.



Obrázok č. 4: Problémy K-means algoritmu s klastrami rôznej hustoty. Vľavo vidíme nesprávne rozdelenie vstupných dát do 3 klastrov. Vpravo vidíme rozdelenie vstupných dát do 5 klastrov, pričom klastre väčšej hustoty sú už určené správne.

Ďalšie problémy s rozdelením vstupných dát do klastrov sa často vyskytujú, ak chceme rozdeliť vstupné dáta do klastrov neguľových tvarov. Obrázok č. 5 zachytáva takýto prípad.



Obrázok č. 5: Problémy K-means algoritmu s klastrami rôznych tvarov. Vľavo vidíme nesprávne rozdelenie vstupných dát do 2 klastrov. Vpravo vidíme rozdelenie vstupných dát do 5 klastrov, pričom jednotlivé klastre už neobsahujú vstupné vzory z inej množiny.

4. Fuzzy c-means (FCM) algoritmus

Fuzzy c-means algoritmus [3] je klastrovacia metóda typu učenia bez učiteľa. Rozdeľuje množinu vstupných vektorov do K zhlukov. Zásadným rozdielom oproti K-means algoritmu (0) je, že zhluky už nemusia tvoriť navzájom disjunktné podmnožiny, čiže vstupný vektor môže zároveň patriť do dvoch alebo viacerých klastrov.

Nech $X = \{\vec{x}_p \in R^I, p = 1, \dots, P\}$ je množina P vstupných vektorov, rozdelených do K zhlukov v_k, $k = 1, \dots, K$. Predpokladá sa $k \ll P$. Zhluk v_k je reprezentovaný fuzzy centroidom $\vec{c}_k, k = 1, \dots, K$. Keďže vstupné vektory môžu patriť súčasne do viacerých zhlukov, má každý vstupný vektor $\vec{x}_p \in X$ priradený stupeň členstva u_{pk} ku k-tému zhluku v_k, pričom $0 \leq u_{pk} \leq 1$. Stupeň príslušnosti vstupných vektorov do jednotlivých zhlukov je určený členskou maticou $U = \left(u_{pk}\right)_{1 \leq p \leq P, 1 \leq k \leq K}$. Počet riadkov členskej matice je určený počtom vstupných vektorov P a počet stĺpcov je určený počtom klastrov K. Parameter m je fuzzyfikačný parameter, pre ktorý platí $1 \leq m < \infty$.

Pre každý vstupný vektor $\vec{x}_p \in X$ platí, že súčet všetkých členských hodnôt u_{pk} tohto vektora k všetkým klastrom je rovný jednej.

$$\forall p | \sum_{k=1}^{K} u_{pk} = 1 \tag{4.1}$$

Pre každý zhluk v_k, $k = 1, \dots, K$ platí, že nie je prázdny ani plný.

$$\forall j | 0 < \sum_{p=1}^{P} u_{pk} < P \tag{4.2}$$

4.1. Chybová funkcia pre FCM algoritmus

Cieľom FCM algoritmu je minimalizovať hodnotu chybovej funkcie E_{FCM}, teda minimalizovať vážené vzdialenosti medzi jednotlivými vstupnými vektormi a príslušnými centroidami. Funkcia E_{FCM} je daná predpisom:

$$E_{FCM} = \sum_{p=1}^{P} \sum_{k=1}^{K} u_{pk}^{m} \left\| \vec{x}_p - \vec{c}_k \right\|^2, \qquad 1 < m < \infty, \tag{4.3}$$

$\left\|\vec{x}_p - \vec{c}_k\right\|$ určuje vzdialenosť medzi vektorom $\vec{x}_p$ a fuzzy centroidom $\vec{c}_k$ v zvolenej metrike.

4.2. FCM algoritmus

Na začiatku algoritmu užívateľ zvolí hodnoty vstupných parametrov. To znamená požadovaný počet klastrov K, hodnotu fuzzyfikačného parametra m a hodnotu ukončovacieho kritéria ε. Náhodne vygenerujeme členskú maticu $U^{(0)}$. Kým nie je splnená ukončovacia podmienka algoritmu, opakovane prepočítavame pozície centroidov a hodnoty prvkov členskej matice podľa nasledujúcich vzťahov:

$$\vec{c}_k(t) = \frac{\sum_{p=1}^{P} u_{pk}^m(t)\vec{x}_p}{\sum_{p=1}^{P} u_{pk}^m(t)} \qquad k = 1, \dots, K \tag{4.4}$$

$$u_{pk}(t+1) = \frac{\left(\frac{1}{\left\|\vec{x}_p - \vec{c}_k(t)\right\|^2}\right)^{\frac{1}{m-1}}}{\sum_{j=1}^{K}\left(\frac{1}{\left\|\vec{x}_p - \vec{c}_j(t)\right\|^2}\right)^{\frac{1}{m-1}}} \tag{4.5}$$

$$p = 1, \dots P, \qquad k = 1, \dots, K$$

Podmienkou k ukončeniu FCM algoritmu je, že maximálna hodnota zmien pozícií centroidov jednotlivých zhlukov medzi jednotlivými iteráciami algoritmu klesne aspoň na hodnotu ukončovacieho kritéria ε [3], [38].

Pseudo-kód: FCM algoritmus

1. Inicializácia:
   Zvoľ parametre K, m, ε a t. Náhodne zvoľ maticu $U^{(0)}$, pre ktorej prvky platí $0 \leq u_{pk} \leq 1$.

2. Aktualizácia pozícií centroidov:
   Vytvor nové centrá fuzzy zhlukov podľa predpisu:

$$\vec{c}_k(t) = \frac{\sum_{p=1}^{P} u_{pk}^m(t)\vec{x}_p}{\sum_{p=1}^{P} u_{pk}^m(t)} \qquad k = 1, \dots, K \tag{4.6}$$

3. Aktualizácia členskej matice:

Vypočítaj aktualizovanú členskú maticu $U^{(t+1)}$ podľa predpisu:

$$u_{pk}(t+1) = \frac{\left(\frac{1}{\left\|\vec{x}_p - \vec{c}_k(t)\right\|^2}\right)^{\frac{1}{m-1}}}{\sum_{j=1}^{K}\left(\frac{1}{\left\|\vec{x}_p - \vec{c}_j(t)\right\|^2}\right)^{\frac{1}{m-1}}} \qquad (4.7)$$

$$p = 1, \dots P, \qquad k = 1, \dots, K$$

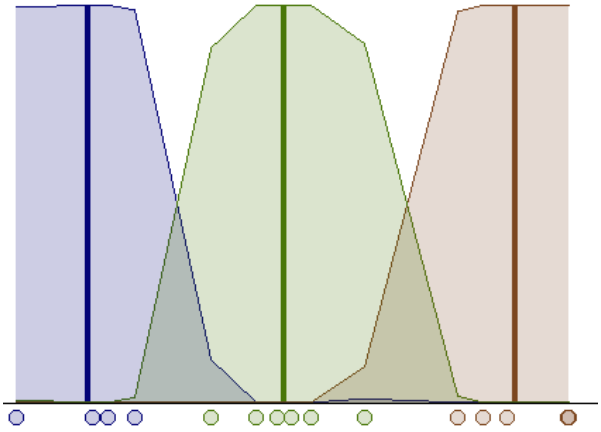
4. Ukončenie algoritmu:
   Vyhodnoť ukončovaciu podmienku.

$$\Delta = \left\|U^{(t+1)} - U^{(t)}\right\| = max_{pk}\left|u_{pk}(t+1) - u_{pk}(t)\right| \qquad (4.8)$$

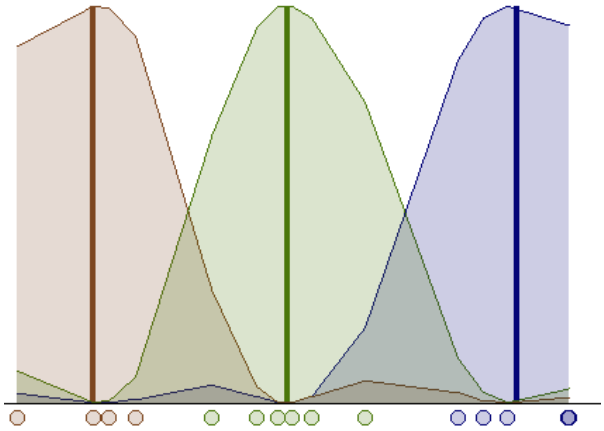
   Ak $\Delta > \varepsilon$, tak nastav $t = t + 1$ a prejdi na krok číslo 2.
   Ak $\Delta \leq \varepsilon$, tak ukonči algoritmus.
   Hodnotu ukončovacieho kritéria volíme z intervalu $0 \leq \varepsilon \leq 1$.

Fuzzyfikačný parameter m má dôležitý vplyv na výkonnosť algoritmu. Určuje, ako veľmi sa mení príslušnosť vstupných vektorov ku klastrom vzhľadom k ich vzdialenosti od centroidov jednotlivých zhlukov. V literatúre bežne máva tento parameter nastavenú hodnotu rovnú dvom. Avšak pomocou experimentov [38], v ktorých sa hľadala optimálna kombinácia hodnoty fuzzyfikačného parametra a počtu klastrov, sa preukázalo, že optimálna hodnota m sa môže líšiť od spomínanej m = 2.0. Na rôznych množinách vstupných dát sa testovali hodnoty parametru $m = \{1.1, 1.5, 2.0, 2.2, 2.7\}$.

Pri hodnotách fuzzyfikačného parametra blízkych jednotke je možnosť súčasného zaradenia vstupných vektorov do viacerých klastrov minimálna. Pri hodnote $m = 1$ algoritmus degraduje na K-means klastrovanie. Naopak pri vyšších hodnotách m vstupné vektory patria súčasne do viacerých klastrov výraznejšou mierou.



Obrázok č. 6: Rozdelenie vstupných dát do 3 klastrov pomocou FCM algoritmu, hodnota fuzzyfikačného parametra $m = 1.5$ [55].



Obrázok č. 7: Rozdelenie vstupných dát do 3 klastrov pomocou FCM algoritmu, hodnota fuzzyfikačného parametra $m = 2$ [55].

4.3. Kritéria pre validitu klastrov

Pri mnohorozmerných vstupných dátach býva ťažké vhodne vizualizovať výsledné klastrovanie. Často sa stáva, že od pohľadu nie je možné posúdiť, ako kvalitné rozdelenie dát do klastrov sme získali. Aby sme boli schopný rozhodnúť, ktoré z výsledných klastrovaní je to najkvalitnejšie, potrebujeme nejaký mechanizmus, ktorý nám umožní ich vzájomné porovnanie. Z tohto dôvodu bolo definovaných viacero kritérií validity klastrov hodnotiacich kvalitu rozdelenia vstupných dát do klastrov . Pri fuzzy zhlukovaní sa najčastejšie využívajú nasledovné validačné kritéria [49]:

Koeficient členskej funkcie

Toto kritérium [6], [47] definujeme predpisom:

$$F_K(U) = \frac{\sum_{p=1}^{P}\sum_{k=1}^{K} u_{pk}^2}{P} \tag{4.9}$$

Toto kritérium naznačuje priemerný počet zdieľaných členstiev medzi dvojicami fuzzy podmnožín v U. Koeficient členskej funkcie môže nadobúdať hodnoty z intervalu $\frac{1}{K} \leq F_K(U) \leq 1$. Optimálne rozdelenie do klastrov je také, keď funkcia $F_K(U)$ nadobúda maximum. Zároveň vieme pomocou tohto koeficientu určiť optimálny počet klastrov, pri ktorom klastrovanie dosiahne najvyššej kvality. Optimálny počet klastrov dostaneme vyriešením vzťahu:

$$\max_{2\leq K\leq P-1}\{\max_U[F_K(U)]\} \tag{4.10}$$

Entropia členskej funkcie

J. Bezdek navrhol toto kritérium [4], [5], [6] a je definované predpisom:

$$H_K(U) = -\frac{\sum_{p=1}^{P}\sum_{k=1}^{K} u_{pk} log_a(u_{pk})}{P} \qquad a \in (1,\infty) \tag{4.11}$$

Entropia členskej funkcie popisuje mieru neurčitosti v rozdelení do fuzzy klastrov, pričom môže nadobúdať hodnoty z intervalu $0 \leq H_K \leq log_a K$. Keďže pri klastrovaní chceme túto mieru neurčitosti minimalizovať, tak optimálne rozdelenie do klastrov je také, keď funkcia $H_K(U)$ nadobúda minimum. Optimálny počet klastrov v tomto prípade vypočítame vyriešením vzťahu:

$$\min_{2\leq K\leq P-1}\{\min_U[H_K(U)]\} \tag{4.12}$$

Entropia $H_K(U)$ je viac citlivá ako koeficient $F_K(U)$ na lokálne zmeny, pričom medzi týmito kritériami platia nasledujúce vzťahy:

$$F_K = 1 \Leftrightarrow H_K = 0 \tag{4.13}$$

$$F_K = \frac{1}{K} \Leftrightarrow H_K = log_a K \tag{4.14}$$

$$\frac{1}{K} \leq F_K \leq 1, \quad 0 \leq H_K \leq log_a K \tag{4.15}$$

Nevýhodou zmienených kritérií je ich často monotónna závislosť na počte klastrov K. Koeficient členskej funkcie môžeme modifikovať, tak aby došlo k redukcii tejto závislosti [2]. Definujme modifikovanú verziu $F'_K(U)$ nasledujúcim spôsobom:

$$F'_K(U) = 1 - \frac{K}{K-1}\left(1 - F_K(U)\right) \tag{4.16}$$

Pre hodnoty, ktoré môže toto modifikované kritérium nadobudnúť platí $0 \leq F'_K(U) \leq 1$. Optimálny počet klastrov získame vyriešením vzťahu:

$$\max_{2 \leq K \leq P-1}\{\max_U [F'_K(U)]\} \tag{4.17}$$

Windhamov proporčný exponent

Toto kritérium navrhol Windham [50] a definujeme ho predpisom:

$$W_K(U) = -\sum_{p=1}^{P} ln\left[\sum_{k=1}^{\lfloor {\mu_p}^{-1} \rfloor} (-1)^{k+1} \binom{K}{k} \left(1 - k.\mu_p\right)^{K-1}\right] \tag{4.18}$$
$$\mu_p = \max_{1 \leq k \leq K}\{u_{kp}\}$$

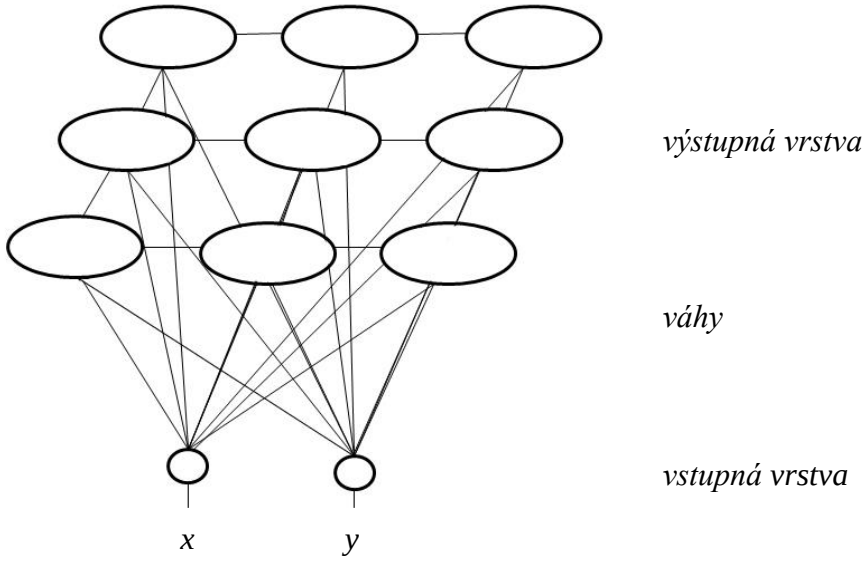
Windham vo svojej práci tvrdí, že je prirodzené zamerať sa na maximálne hodnoty prvkov uvedených v stĺpcoch členskej matice U. Veľké hodnoty maximálnych prvkov totižto predznamenávajú jasnú príslušnosť niektorých vstupných vektorov do nejakých zhlukov. To nám signalizuje identifikáciu určitej vnútornej štruktúry vo vstupnom priestore. Optimálne rozdelenie do klastrov je také, keď funkcia $W_K(U)$ nadobúda maximum. Optimálny počet klastrov v tomto prípade vypočítame vyriešením vzťahu:

$$\max_{2 \leq K \leq P-1}\{\max_U [W_K(U)]\} \tag{4.19}$$

5. Kohonenové mapy

Medzi základné typy neurónových sieti patria samoorganizačné neurónové siete, tzv. SOM – Self-Organizing Map. Už ako názov napomína, jedná sa o siete, ktorých učiaci proces je typu učenia bez učiteľa. SOM sa využívajú na mapovanie vstupného príznakového priestoru, v ktorom sa hľadajú oblasti so vzormi, ktoré majú navzájom podobné vlastnosti a budú vo výsledku vytvárať zhluky. Pri mapovaní by sa malo zachovávať topologické usporiadanie vstupných vzorov.

Kohonenové mapy navrhnuté T. Kohonenom v roku 1982 sú jedným z hlavným predstaviteľom neurónových sieti typu SOM. Základná myšlienka pre ich vznik pochádza z neurobiológie, kde sa zistilo, že väčšina štruktúr neurónov v mozgu má lineárnu alebo rovinnú topografickú anatómiu napriek tomu, že zmyslové vnemy bývajú mnohorozmerné [24]. Toto je dôležitá vlastnosť, ktorá sa preniesla aj do Kohonenových máp. Použitie Kohonenových máp je teda veľmi výhodné, ak chceme vizualizovať mnohorozmerné dáta do nízkodimenzionálneho priestoru.



Obrázok č. 8: Príklad Kohonenovej mapy s usporiadaním neurónov do topologickej mriežky so štvorcovým okolím a s dvoma vstupnými neurónmi.

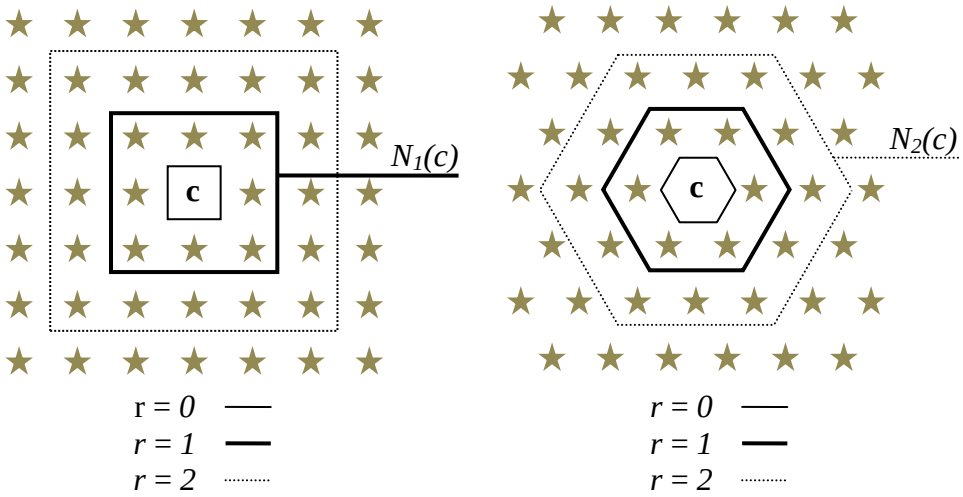
5.1. Topológia Kohonenovej mapy

Jedná sa o dvojvrstvovú neurónovú sieť (vstupná a výstupná vrstva), pričom pre každý vstupný neurón platí, že je spojený synaptickou váhou s každým neurónom vo výstupnej vrstve. Neuróny vo výstupnej vrstve sú usporiadané do určitej topologickej štruktúry. Najčastejšie sa používa dvojrozmerná mriežka, prípadne jednorozmerná rada. Voľba konkrétnej topologickej štruktúry je dôležitá v procese učenia, pretože nám zároveň určuje, ktoré neuróny spolu susedia. Počas učenia sa aktualizuje pozícia víťazného neurónu a susedných neurónov z okolia víťazného neurónu. Pod pojmom okolie neurónu c budeme rozumieť množinu neurónov $N_r(c)$ spĺňajúcich podmienku, že ich topologická vzdialenosť na topologickej mriežke od neurónu c je menšia alebo rovná hodnote polomeru r.

$$N_r(c) = \{j, d(j,c) \leq r\} \tag{5.1}$$

Vzdialenosť neurónov $d(j,c)$ je práve závislá na topologickej štruktúre výstupných neurónov. Vypočítame ju ako počet hrán na najkratšej ceste z neurónu c do neurónu j. Veľkosť okolia sa v procese učenia s časom mení. Na začiatku algoritmu sa zvolí veľký polomer okolia r, napr. polovica veľkosti mriežky. Postupne sa veľkosť okolia s narastajúcim počtom krokov učenia zmenšuje. Na konci učiaceho procesu by malo okolie zahŕňať už len samotný neurón c. Tento pokles veľkosti okolia spôsobuje, že na predložený vstupný vektor sa postupne adaptuje stále menšie množstvo neurónov, ustália sa hodnoty váh, a tým sa zvyšuje presnosť výslednej Kohonenovej mapy.

V blízkosti okrajov mapy (rada, mriežka, ...) okolie neurónov nie je symetrické, z tohto dôvodu bývajú okrajové časti mierne kontrahované smerom do vnútra mapy. Toto môžeme odstrániť tak, že algoritmicky prepojíme odpovedajúce si okraje mapy do slučky.



Obrázok č. 9: Príklad okolia neurónu c s hodnotu polomeru $r = \{1, 2\}$ vo štvorcovej a hexagonálnej topografickej štruktúre výstupných neurónov.

5.2. Funkcia laterálnej interakcie

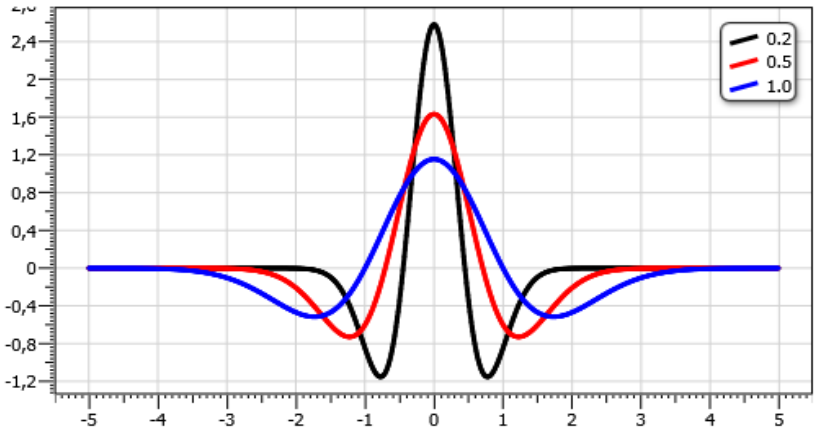
Funkcia laterálnej interakcie určuje mieru a druh vzájomnej interakcie (inhibícia, excitácia) medzi neurónmi v topologickej mriežke v závislosti na topologickej vzdialenosti neurónov j od neurónu c. Funkciu laterálnej interakcie budeme označovať $\phi_c(j)$. Uvedieme si dve funkcie, ktoré je možno zvoliť za funkciu laterálnej interakcie:

Funkcia typu mexického klobúka

Pre túto funkciu platí, že laterálna interakcia má excitačný vplyv na blízke neuróny a v menšej miere inhibičný vplyv pre navzájom vzdialenejšie neuróny. S narastajúcou vzdialenosťou neurónov inhibičný vplyv klesá k nule. Funkcia typu mexického klobúka je definovaná predpisom:

$$\phi_c(j) = \frac{2}{\sqrt{3\sigma}\pi^{\frac{1}{4}}}\left(1 - \frac{d(j,c)^2}{\sigma^2}\right)e^{-\frac{d(j,c)^2}{2\sigma^2}}, \qquad \sigma > 0 \qquad (5.2)$$

kde σ je šírka funkcie.



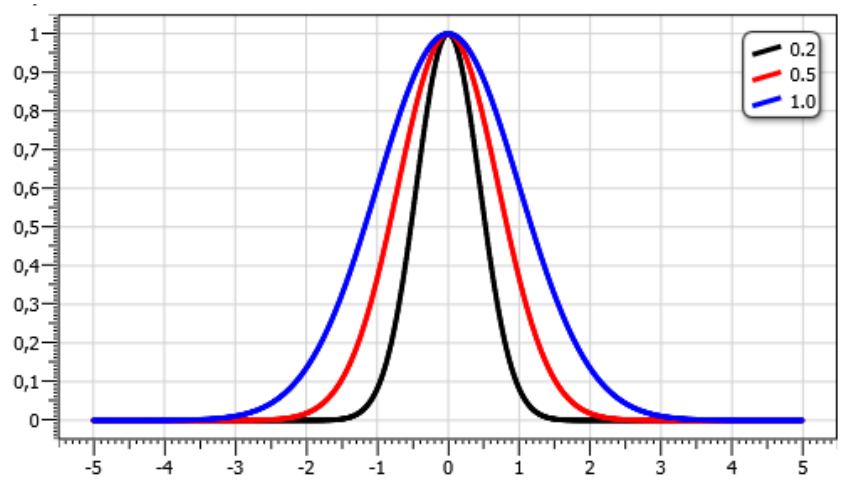
Obrázok č. 10: Priebeh funkcie typu mexického klobúka so stredom $c = 0$ s rôznou hodnotou parametra šírky $\sigma, \sigma \in \{0.2, 0.5, 1.0\}$.

Gaussova funkcia

Gaussova funkcia je definovaná predpisom:

$$\phi_c(j) = a * \exp\left(-\frac{(j-c)^2}{2\sigma^2}\right), \qquad \sigma > 0 \tag{5.3}$$

Parameter c určuje pozíciu stredu Gaussovej funkcie, parameter σ určuje šírku Gaussovej funkcie. Parameter šírky σ ovplyvňuje strmosť tejto funkcie. Parameter a ovplyvňuje výšku vrcholu funkcie. Graf tejto funkcie pripomína tvar zvončeka.



Obrázok č. 11. Priebeh Gaussovej funkcie so stredom $c = 0$ s rôznou hodnotou parametra šírky $\sigma, \sigma \in \{0.2, 0.5, 1.0\}$. Parameter $a = 1$. Za funkciu laterálnej interakcie sa často volí práve Gaussova funkcia.

5.3. Algoritmus učenia Kohonenovej mapy

Nech $X = \{\vec{x}_p \in R^I, p = 1, \dots, P\}$ je množina P trénovacích vzorov. Parameter učenia η(t), $0 \leq \eta(t) \leq 1$ ovplyvňuje rýchlosť učenia siete. Na začiatku volíme vyššiu hodnotu blízku jednej, ktorá počas procesu učenia siete klesá k nule. Parameter učenia by mal splňovať požiadavky:

$$\sum_{t=1}^{\infty} \eta(t) = \infty \tag{5.4}$$

$$\sum_{t=1}^{\infty} \eta^2(t) < \infty \tag{5.5}$$

Ďalej w_{ij}, $0 \leq i \leq I-1, 0 \leq j \leq O-1$ určuje váhu synapsie zo vstupného neurónu i k výstupnému neurónu j, pričom I (input) určuje počet vstupných neurónov a O (output) určuje počet výstupných neurónov.

Na začiatku algoritmu učenia Kohonenovej mapy zvolíme počiatočne hodnoty parametrov, a to hodnotu parametra učenia $\eta(t)$, polomer r okolia neurónov vo výstupnej vrstve siete a váhy siete $w_{ij}, 0 \leq i \leq I-1, 0 \leq j \leq O-1$ nastavíme na malé náhodné hodnoty.

Náhodne vyberieme trénovací vzor $\vec{x} \in X$ a ten predložíme na vstup sieti. Tento vzor $\vec{x}$ porovnáme s váhovými vektormi $\vec{w}_j$ všetkých výstupných neurónov a vyberieme taký neurón c, ktorého váhový vektor je najbližšie (je najpodobnejší) predloženému vzoru. Hovoríme, že tento neurón c vyhral kompetíciu a označíme ho ako víťazný neurón. Takýto princíp sa nazýva „víťaz berie všetko“ (“winner takes all”).

Ďalej chceme, aby víťazný neurón c a neuróny z jeho okolia $N_r(c)$ ešte lepšie reprezentovali predložený vzor $\vec{x}$. To dosiahneme posunutím týchto neurónov bližšie k predloženému vzoru $\vec{x}$. V tomto bode sa práve využije funkcia laterálnej interakcie (5.2), ktorá spôsobí, že neuróny blízko víťazného neurónu c sa posunú spolu s víťazným neurónom c bližšie k predloženému vzoru $\vec{x}$ (stanú sa citlivejšími na predložený vzor a vzory jemu podobné). Vzdialenejšie neuróny z okolia $N_r(c)$ sa naopak ešte viac vzdialia od víťazného neurónu c (ich reakcia bude utlmená). To, ktoré váhy sa budú aktualizovať, záleží na veľkosti okolia víťazného neurónu $N_r(c)$.

S narastajúcim počtom iterácií klesá veľkosť okolia, a teda klesá aj počet neurónov, ktorých sa tento posun týka. Následne pokračujeme predložením ďalšieho trénovacieho vzoru $\vec{x} \in X$ na vstup sieti. Tento postup opakujeme, kým nedosiahneme požadovaný počet iterácií.

Pseudo-kód: Učenie Kohonenovej mapy

1. Inicializácia:
   Zvoľ náhodné malé hodnoty váh $w_{ij},\ 0 \leq i \leq I-1,\ 0 \leq j \leq O-1$ medzi I vstupnými a O výstupnými neurónmi. Ďalej zvoľ počiatočný polomer r okolia, parameter učenia $\eta(t)$ a funkciu laterálnej interakcie $\phi(c,j)$.
   Polomer okolia r volíme na začiatku veľký, napr. polovica veľkosti mriežky. Parameter učenia $\eta(t)$, $0 \leq \eta(t) \leq 1$ nastavíme na hodnotu blízko jednej.

2. Predloženie vzoru:
   Náhodne vyber trénovací vzor $\vec{x} \in X$ a ten predlož na vstup neurónovej sieti.

3. Výpočet vzdialenosti:
   Spočítaj vzdialenosti d_j medzi trénovacím vzorom $\vec{x}$ a váhovým vektorom $\vec{w}_j$ pre každý výstupný neurón $j, 0 \leq j \leq O-1$ podľa predpisu:

$$d_j = \sqrt{\sum_{i=0}^{I-1} \left(\vec{x}_i(t) - \vec{w}_j(t)\right)^2}, \qquad 0 \leq j \leq O-1 \tag{5.6}$$

4. Výber víťazného neurónu:
   Nájdi najbližší výstupný neurón c, ktorý má minimálnu hodnotu d_j a označ ho ako víťaza.

$$c = \arg\min_j \left(d_j\right), \qquad 0 \leq j \leq O-1 \tag{5.7}$$

5. Aktualizácia váh:
   Aktualizuj váhy pre víťazný neurón c a pre všetky neuróny z okolia definovaného pomocou $N_c(r)$ podľa predpisu:

$$\vec{w}_j(t+1) = \begin{cases} \vec{w}_j(t) + \eta(t)\phi(c,j)\left(\vec{x}_i(t) - \vec{w}_j(t)\right), & j \in N_c(r) \\ \vec{w}_j(t), & j \notin N_c(r) \end{cases} \tag{5.8}$$

6. Ukončenie algoritmu:
   Ak bol dosiahnutý požadovaný počet iterácií, tak ukonči algoritmus.
   V opačnom prípade aktualizuj hodnoty parametra učenia $\eta(t)$ a polomeru r. Hodnotu parametra učenia aktualizuj podľa:

$$\eta(t) = 0.9\left(1 - \frac{t}{N_{iter}}\right) \tag{5.9}$$

N_{iter} predstavuje užívateľom zadaný počet krokov učiaceho procesu. Veľkosť polomeru okolia by mala s narastajúcim časom t klesnúť až na nulu, napríklad lineárne. Pokračuj na krok algoritmu číslo 2.

V piatom kroku algoritmu sa pomerne často za funkciu laterálnej interakcie (5.2) volí Gaussova funkcia so stredom v c, ktorá viac odpovedá biologickým interakciam. Aktualizácia váh v tomto prípade prebieha podľa predpisu:

$$\vec{w}_j(t+1) = \vec{w}_j(t) + \\ + \eta(t)\exp\left(-\frac{d(j,c)^2}{2\sigma^2(t)}\right)\left(\vec{x}_i(t) - \vec{w}_j(t)\right), \tag{5.10}$$

kde $\sigma(t)$ je šírka funkcie. Platí, že hodnoty parametrov $\sigma(t)$ a $\eta(t)$ sa s narastajúcim časom t zmenšujú. Časová náročnosť tohto riešenia je však vyššia, pretože v každom kroku prechádzame a aktualizujem všetky váhové vektory $\vec{w}_j$, $0 \leq j \leq O-1$ v sieti.

Celý proces učenia Kohonenovej mapy sa dá rozdeliť na dve fázy, hrubé učenie a dolaďovanie [24].

Pod hrubým učením rozumejme napríklad prvých tisíc krokov učiaceho procesu. V tejto fáze majú parameter učenia $\eta(t)$ a veľkosť okolia $N_r(c)$ veľké hodnoty. Parameter učenia $\eta(t)$ začína s hodnotou blízkou jednotke a postupne klesá. Experimentálne sa preukázalo, že nie je dôležité, či tento parameter klesá lineárne, exponenciálne, alebo iným spôsobom. Parameter učenia η(t) by mal počas hrubého učenia klesnúť na hodnotu približne 0.02. V ďalšej fáze pokračuje pokles tohto parametru lineárne k nule.

Hodnota polomeru r okolia neurónov by sa mala počas fázy hrubého učenia meniť nasledovne. Na začiatku môžeme zvoliť za polomer r polovicu rozmeru mapy. Tento parameter môže lineárne klesať až na hodnotu jedna. V dolaďovacej fáze môžu byť ešte zo začiatku do okolia $N_r(c)$ zahrnuté najbližšie neuróny, pričom hodnota polomeru r klesne až na nulu a ďalej sa bude už upravovať len víťazný neurón c. Počas hrubého učenia dôjde k rozmiestneniu výstupných neurónov približne do oblastí, kde sa vyskytujú vstupné vzory. Pozície neurónov sa v ďalšej fáze menia už len pomerne málo.

Výsledná presnosť Kohonenovej mapy záleží na počte vykonaných iterácií učiaceho algoritmu, predovšetkým v záverečnej dolaďovacej fáze. Ta by mala byť primerane dlhá. V publikáciách [24], [25] sa uvádza, že počet iterácií by mal byť najmenej päťstokrát väčší ako je počet výstupných neurónov v sieti. Bežne sa necháva učiaci

algoritmus vykonať až do 100 tisíc iterácií, ale vždy záleží na konkrétnom druhu úlohy. Napríklad pri úlohe rozpoznávania rečí 10 tisíc iterácií môže byť dostatočné množstvo.

Uvedený algoritmus učenia Kohonenovej mapy (5.3) nemusí vždy fungovať podľa našich predstáv. Problém môže nastať v prípade, že jeden alebo malá skupina neurónov často víťazia v kompetícii. Takáto situácia môže vyzerať napríklad nasledovne. Vstupné dáta nech sú rozdelené do dvoch izolovaných od seba dostatočne vzdialených oblasti. Inicializačné rozmiestnenie výstupných neurónov sa volí náhodne, takže sa môže stať, že všetky tieto neuróny sa vyskytnú v jednej z týchto oblasti. Keď predložíme vzor z druhej oblasti, tento vzor pritiahne jeden neurón z prvej oblasti bližšie k sebe. Potom tento neurón už bude vždy víťaziť v kompetícii pre vstupy z druhej oblasti a táto oblasť bude reprezentovaná len jedným neurónom.

Odstrániť tento problém môžeme nasledujúcim spôsobom. Pre každý výstupný neurón si budeme pamätať počet jeho doposiaľ dosiahnutých víťazstiev v kompetícii. Ak v kompetícii zvíťazí neurón, ktorého počet víťazstiev je príliš veľký, vyradíme tento neurón na určitú dobu z kompetície a necháme zvíťaziť namiesto neho iný výstupný neurón. Tento postup sa bežne nazýva ako princíp svedomia.

5.4. Aplikácie Kohonenových máp

Aplikácií Kohonenových máp je veľmi veľa. Jednou z najznámejších aplikácií je rozpoznávanie hovorenej reči. Na vstup predložíme nejaký zvukový záznam a ako výstup dostaneme textový prepis tohto zvukového záznamu. Základným impulzom pre vznik takéhoto prístroja bolo odstránenie ručného písania textu. Človek jednoducho diktuje text a stroj ho za neho zapisuje. Nevýhodou tohto stroja je naviazanosť na konkrétnu osobu, s ktorou sa učil rozpoznávať jednotlivé písmena a slová.

Rozsiahly zoznam možných aplikácií Kohonenových máp je uvedený v [24]. Spomeniem niekoľko z nich, ktoré má najviac zaujali. Jednou z oblastí je robotika, kde sa Kohonenové mapy využívajú pri navigácii robotov a na ich samotnú kontrolu [18]. Jednou z hlavných úloh robotiky je umiestnenie ramena robota do požadovanej polohy [31]. Aby sa roboti mohli pohybovať v priestore musia spracovávať množstvo informácií zo svojho okolia ako napríklad obraz. A práve na spracovanie obrazu sa opäť môžu využiť Kohonenové mapy. Medzi úlohy z tejto oblasti patrí napríklad digitálne zakódovanie obrazu [1], kompresia obrazu [29] a segmentácia obrazu [23].

V medicíne sa s Kohonenovými mapami môžeme stretnúť pri analýze signálov z elektrokardiogramu [34], pri analýze zvukov, ktoré vydávajú pľúca [21], pri analýze výsledkov hladiny glukózy v krvi získaných pomocou glukometra [27] a pri množstve ďalších iných aplikácií z rôznych oblastí.

6. Varianty Kohonenových máp s adaptatívnou topológiou

Klasické Kohonenové mapy reprezentujeme pomocou dvojvrstvovej neurónovej siete (vstupná a výstupná vrstva). Každý vstupný neurón je spojený s každým neurónom vo výstupnej vrstve. Neuróny vo výstupnej vrstve sú usporiadané do určitej topologickej štruktúry.

Tvar tejto topologickej štruktúry musí byť užívateľom zadaný dopredu ešte pred začiatkom algoritmu učenia Kohonenovej mapy (5.3). Tvar a veľkosť zvolenej topologickej štruktúry v ďalšom priebehu algoritmu učenia zostávajú nemenné. Práve toto je veľmi obmedzujúce pri použití Kohonenových máp. Kohonenové mapy sa využívajú na mapovanie mnohorozmerného vstupného priestoru do nízkodimenzionálneho priestoru. Toto mapovanie je teda už dopredu obmedzované veľkosťou a tvarom zvolenej topologickej štruktúry výstupných neurónov. Z tohto dôvodu sa začali vyvíjať nové varianty Kohonenových máp, ktoré umožňujú počas učiaceho procesu meniť nielen pozíciu a počet neurónov, ale aj topologickú štruktúru siete.

V ďalšom texte si predstavíme dve varianty tohto typu modelu, a to Kohonenové mapy s rastúcou mriežkou (6.1) a model rastúcich neurónových plynov (0).

6.1. Kohonenové mapy s rastúcou mriežkou

V tomto modeli Kohonenových máp [10] budeme pracovať s usporiadaním neurónov vo vstupnom priestore do topologickej mriežky so štvorcovým okolím o rozmeroch $O_1 \times O_2 = O$. Množinu neurónov topologickej mriežky označme A.

Nech $X = \{\vec{x}_p \in R^I, p = 1, \dots, P\}$ je množina P trénovacích vzorov. Počet neurónov označíme O. Ďalej $\vec{w}_j,\ 0 \leq j \leq O-1$ je váhový vektor neurónu j určujúci jeho pozíciu vo vstupnom priestore. Parameter $\tau_j, 0 \leq j \leq O-1$ udržiava informáciu o doposiaľ dosiahnutom počte víťazstiev v kompetícii pre každý neurón. Parameter λ_r (r-rast) ovplyvňuje rýchlosť zväčšovania počtu neurónov vo vstupnom priestore. Parameter učenia η_0 má konštantnú hodnotu.

6.1.1. Algoritmus učenia Kohonenovej mapy s rastúcou mriežkou

Na začiatku algoritmu učenia sa začína s mriežkou malých rozmerov (napr. 2×2). Zvolíme počiatočne hodnoty parametrov, a to hodnotu parametra učenia η_0 a parametra λ_r. Hodnoty všetkých parametrov τ_j nastavíme na nulu, $\tau_j = 0 | \forall j$.

Náhodne zvolíme trénovací vzor $\vec{x} \in X$ a ten predložíme na vstup sieti. V zvolenej metrike vzor $\vec{x}$ porovnáme s váhovými vektormi $\vec{w}_j$ všetkých neurónov a vyberieme taký neurón c, ktorého váhový vektor je najbližšie (je najpodobnejší) trénovaciemu vzoru. Hovoríme, že tento neurón vyhral kompetíciu a označíme ho ako víťazný neurón c. Aktualizujeme hodnoty všetkých váhových vektorov $\vec{w}_j$ podľa vzťahu

$$\vec{w}_j = \vec{w}_j + \eta_0 \exp\left(-\frac{d(j,c)^2}{2\sigma^2}\right)\left(\vec{x} - \vec{w}_j\right) \quad \forall j \in A \tag{6.1}$$

Za funkciu laterálnej interakcie je v tomto prípade zvolená už spomínaná Gaussova funkcia (5.2). Víťaznému neurónu c zvýšime počet víťazstiev τ_c o jedna. V tomto okamihu dochádza k rozhodnutiu, či sa do topologickej mriežky pridajú ďalšie neuróny, alebo sa pokračuje predložením ďalšieho trénovacieho vzoru.

Ak počet doposiaľ vykonaných iterácií učiaceho algoritmu prevýšil počet $O_1 \times O_2 \times \lambda_r$, dôjde k pridaniu nových neurónov nasledujúcim spôsobom. Podľa najväčšej hodnoty τ_j, $0 \leq j \leq O - 1$ nájdeme výstupný neurón q s najväčším počtom víťazstiev v kompetícii. Práve do okolia neurónu q budeme vkladať nové neuróny, čím zvýšime hustotu neurónov v danej oblasti a dosiahneme tým presnejšie a rovnomernejšie rozdelenie neurónov vo vstupnom priestore. Nájdeme k neurónu q susedný neurón f taký, že váhové vektory týchto neurónov sú si najviac vzdialené, a teda najmenej podobné. Hľadáme len medzi priamymi susedmi, tí môžu byť maximálne štyria.

Do topologickej mriežky vložíme riadok (prípadne stĺpec) nových neurónov, a to medzi riadky (prípadne stĺpce), ktoré obsahujú neuróny q a f. Bez ujmy na všeobecnosti môžeme predpokladať, že neuróny q a f sa nachádzajú na r-tom riadku mriežky A. Konkrétne nech $q = a_{rs}, f = a_{rs+1}$. Medzi stĺpce s a $s + 1$ vložíme nový stĺpec s' s k novými neurónmi. Váhové vektory pre nové neuróny sa vytvoria podľa vzťahu:

$$\vec{w}_{rs'} = \frac{\vec{w}_{rs} + \vec{w}_{rs+1}}{2} \qquad 1 \leq r \leq O_1 \tag{6.2}$$

Zvýšime počet stĺpcov mriežky O_2 a všetky premenné τ_j opäť nastavíme na nulu, $\tau_j = 0 | \forall j$. Ak nebola splnená ukončovacia podmienka algoritmu, pokračujeme s predložením ďalšieho trénovacieho vzoru $\vec{x} \in X$.

Ukončovacia podmienka algoritmu je potrebná, aby sme zabránili nekontrolovanému zväčšovaniu počtu neurónov vo vstupnom priestore. Možnosti je viacero, medzi

najjednoduchšie patrí napríklad určenie maximálneho počtu neurónov, ktoré sa môžu objaviť vo vstupnom priestore. Pri voľbe iných podmienok treba dávať pozor, aby došlo k ich naplneniu.

Po skončení algoritmu (rastovej fázy) sa môže previesť ešte dolaďovacia fáza, počas ktorej dôjde k poslednému jemnejšiemu posunu neurónov. Parameter λ_d (d-doladenie) vyjadruje, koľko sa priemerne vykoná aktualizačných krokov váhových vektorov každého neurónu v dolaďovacej fáze. Parameter učenia $\eta(t)$ na rozdiel od parametra η_0 v rastovej fáze už nemá konštantnú hodnotu, ale klesá s narastajúcim časom t. Za hodnotu parametru η_1 sa volí malá hodnota $\eta_1 < \eta_0$.

Adaptácia váhových vektorov prebieha podľa vzťahu (6.1), pričom priebeh parametra učenia je daný predpisom:

$$\eta(t) = \eta_0 \left(\frac{\eta_1}{\eta_0}\right)^{\frac{t}{t_{\max}}} \qquad t = 1, \dots, t_{\max} \tag{6.3}$$

$$t_{\max} = O_1 \times O_2 \times \lambda_d \tag{6.4}$$

Pseudo-kód: Učenie Kohonenovej mapy s rastúcou mriežkou

1. Inicializácia:
   Začni s topologickou mriežkou malých rozmerov $k \times m$ (napr. 2×2). Inicializuj zložky váhových vektorov $\vec{w}_j$, $0 \leq j \leq O - 1$ malými náhodnými hodnotami, kde O je počet neurónov (v tomto prípade $O = 4$). Nastav hodnoty parametrov $\tau_j = 0,\ 0 \leq j \leq O - 1$.

2. Predloženie trénovacieho vzoru:
   Náhodne vyber trénovací vzor $\vec{x} \in X$ a ten predlož na vstup sieti.

3. Výber víťazného neurónu:
   Nájdi neurón c, ktorého váhový vektor je najbližšie k predloženému trénovaciemu vzoru $\vec{x}$ a označ ho ako víťaza.

   $$c = \arg\min_j \left(\left\|\vec{x} - \vec{w}_j\right\|\right), \qquad \forall j \in A \tag{6.5}$$

   Zvýš hodnotu parametra τ_c označujúceho počet víťazstiev pre neurón c, $\tau_c = \tau_c + 1$.

4. Aktualizácia váh:
   Za funkciu laterálnej interakcie je zvolená Gaussova funkcia (5.2). Aktualizuj váhy podľa predpisu:

$$\vec{w}_j = \vec{w}_j + \eta_0 \exp\left(-\frac{d(j,c)^2}{2\sigma^2}\right)(\vec{x} - \vec{w}_j) \quad \forall j \in A \tag{6.6}$$

Pre výpočet vzdialenosti $d(j,c)$ medzi neurónmi je zvolená Manhattanska metrika definovaná predpisom:

$$d_M(\vec{a}, \vec{b}) = \sum_{i=1}^{n} |a_i - b_i| \tag{6.7}$$

$\vec{a}, \vec{b}$ sú n-prvkové vektory.

Hodnoty parametra učenia η_0 a parametra šírky σ sú konštantné počas celého algoritmu.

5. Rozhodnutie o pridaní neurónov:
   Ak počet doposiaľ vykonaných iterácií algoritmu je rovný $O_1 \times O_2 \times \lambda_r$, pokračuj na krok algoritmu číslo 6. V opačnom prípade pokračuj na krok algoritmu číslo 2.

6. Určenie najčastejšie vyhrávajúceho neurónu:
   Nájdi najčastejšie vyhrávajúci neurón q podľa najväčšej hodnoty τ.

$$q = \arg\max_j (\tau_j) \quad \forall j \in A \tag{6.8}$$

7. Určenie najvzdialenejšieho susedného neurónu:
   Definujme množinu priamych susedov neurónu q predpisom:

$$N_1(q) = \{j, d(q,j) = 1\} \tag{6.9}$$

   Z množiny $N_1(q)$ vyber susedný neurón f, ktorého váhový vektor $\vec{w}_f$ je najvzdialenejší od váhového vektora $\vec{w}_q$.

$$f = \arg\max_{j'} \left(\left\|\vec{w}_{j'} - \vec{w}_q\right\|\right), \ \forall j' \in N_1(q) \tag{6.10}$$

   Medzi riadky (prípadne stĺpce) obsahujúce neuróny q a f sa bude vkladať riadok (prípadne stĺpec) nových neurónov.

8. Pridanie nových neurónov:
   Bez ujmy na všeobecnosti predpokladajme, že neuróny q a f sa nachádzajú na r-tom riadku mriežky A, konkrétne nech $q = a_{rs}, f = a_{rs+1}$. Medzi stĺpce s a $s+1$ vlož nový stĺpec s' s O_1 novými neurónmi. Váhové vektory pre nové neuróny vytvor podľa predpisu:

$$\vec{w}_{rs'} = \frac{\vec{w}_{rs} + \vec{w}_{rs+1}}{2} \qquad 1 \leq r \leq k \tag{6.11}$$

Aktualizuj počet stĺpcov $O_2 = O_2 + 1$.
V prípade vkladania riadku nových neurónov do topologickej mriežky postupujeme podobne. Nastav hodnoty premenných τ_j na nulu, $\tau_j = 0 | \forall j$.

9. Ukončenie algoritmu:
   Ak nie je splnená ukončovacia podmienka pokračuj na krok algoritmu číslo 2.

Nevýhodou tejto varianty Kohonenových máp je, že sa namiesto jedného neurónu vkladá do mriežky naraz väčší počet neurónov. Vždy sa vloží do mriežky celý riadok (prípadne stĺpec) nových neurónov, pričom sa môže stať, že nie všetky vložené neuróny budú plne využité. Tento spôsob vkladania neurónov je zvolený z dôvodu zachovania tvaru topológie neurónov vo vstupnom priestore.

6.2. Model rastúcich neurónových plynov

V tomto modeli [9] na rozdiel od Kohonenových máp s rastúcou mriežkou (6.1) nie sú kladené žiadne požiadavky na tvar alebo veľkosť topologickej mriežky. Keďže počas učiaceho procesu dochádza k postupnému pridávaniu jednotlivých neurónov do vstupného priestoru, výsledná topológia sa tvaruje v priebehu učenia. Množinu neurónov vo vstupnom priestore opäť označme ako A. Tento model je založený na princípe kompetičného Hebbovského učenia.

Nech $X = \{\vec{x}_p \in R^I, p = 1, \dots, P\}$ je množina P trénovacích vzorov. Počet výstupných neurónov označíme O. Niektoré neuróny sú navzájom spojené hranou, množinu týchto hrán označme ako $E, E \subseteq O \times O$. Každá hrana má priradený parameter $vek \in N \cup \{0\}$ vyjadrujúci vek tejto hrany. Ak vek nejakej hrany presiahne hodnotu parametra $vek_{\max}$ určujúceho maximálny povolený vek, tak sa táto hrana odstráni. Pre každý víťazný neurón c, si udržiavame hodnotu lokálnej chyby $Error_c$ vyjadrujúcu súčet kvadrátov vzdialenosti váhového vektora $\vec{w}_c$ víťazného neurónu od predložených trénovacích vzorov $\vec{x}$. Parametre α a β ovplyvňujú zmenu lokálnych chýb jednotlivých neurónov, pričom platí $0 < \alpha, \beta < 1$. Parameter λ určuje počet iterácií algoritmu, ktoré sa musia vykonať, kým dôjde k pridaniu ďalšieho neurónu. Parametre ϵ_b a ϵ_n ovplyvňujú veľkosť priblíženia víťazného neurónu a jeho priamych susedov k predloženému trénovaciemu vzoru $\vec{x}$. Pre tieto parametre platí $1 > \epsilon_b \gg \epsilon_n$. Ďalej $\vec{w}_j$, $0 \leq j \leq O - 1$ je váhový vektor neurónu j určujúci jeho pozíciu vo vstupnom priestore.

6.2.1. Algoritmus učenia rastúcich neurónových plynov

Algoritmus začína s dvoma neurónmi a_1, a_2, ktoré náhodne rozmiestnime vo vstupnom priestore. Zatiaľ nie sú žiadne neuróny v sieti spojené hranou, $E = \emptyset$. Kým nie je splnená ukončovacia podmienka algoritmu, opakujeme nasledujúci postup.

Náhodne zvolíme trénovací vzor $\vec{x} \in X$ a ten predložíme na vstup sieti. V zvolenej metrike vzor $\vec{x}$ porovnáme s váhovými vektormi $\vec{w}_j$ všetkých neurónov a vyberieme také neuróny c_1 a c_2, ktorých váhové vektory sú dva najbližšie (najpodobnejšie) trénovaciemu vzoru. Najbližšiemu neurónu c_1 nastavíme hodnotu lokálnej chyby $Error(c_1)$ podľa vzťahu:

$$Error(c_1) = Error(c_1) + \left\|\vec{w}_{c_1} - \vec{x}\right\|^2 \tag{6.12}$$

Ak medzi neurónmi c_1 a c_2 nevedie hrana, tak ju vytvoríme. Aktualizujeme hodnoty váhových vektorov víťazného neurónu c_1 a jeho priamych susedov podľa vzťahov:

$$\vec{w}_{c_1} = \vec{w}_{c_1} + \epsilon_b\left(\vec{x} - \vec{w}_{c_1}\right)$$
$$\vec{w}_j = \vec{w}_j + \epsilon_n\left(\vec{x} - \vec{w}_j\right) \qquad \forall j \in N_1(c_1) \tag{6.13}$$

Všetkým hranám vychádzajúcim z c_1 zvýšime vek, $vek = vek + 1$. Ak medzi neurónmi c_1 a c_2 vedie hrana, nastavíme jej vek na nulu. V tomto okamihu dochádza k rozhodnutiu, či sa pridá nový neurón do topologickej mriežky vo vstupnom priestore, alebo sa pokračuje predložením ďalšieho trénovacieho vzoru $\vec{x} \in X$.

Ak počet doposiaľ vykonaných iterácií učiaceho algoritmu je násobkom parametra λ, dôjde k pridaniu nového neurónu nasledujúcim spôsobom. Nájdi neurón q s najväčšou lokálnou chybou a z pomedzi priamych susedov neurónu q nájdi neurón f opäť s najväčšou lokálnou chybou. Do stredu hrany spájajúcej neuróny q a f vlož nový neurón r. Pre tento neurón vytvor váhový vektor w_r podľa vzťahu:

$$\vec{w}_r = \frac{\vec{w}_q + \vec{w}_f}{2} \tag{6.14}$$

Odstráň hranu medzi neurónmi q, f a pridaj dve nové hrany medzi neuróny q, r a r, f. Neurónom q a f aktualizuj hodnotu lokálnej chyby vynásobením parametrom α, $0 < \alpha < 1$. Novému neurónu r priraď hodnotu lokálnej chyby neurónu q. Následne aktualizuj hodnoty lokálnych chýb všetkých neurónov vynásobením parametrom $\beta, 0 < \beta < 1$. Ak nebola splnená ukončovacia podmienka algoritmu, pokračujeme s ďalším trénovacím vzorom $\vec{x} \in X$.

Ukončovacia podmienka algoritmu je potrebná, aby sme zabránili nekontrolovanému zväčšovaniu počtu neurónov vo vstupnom priestore. Opäť môžeme použiť napríklad podmienku určujúcu maximálny počet neurónov, ktoré sa môžu objaviť vo vstupnom priestore. Pri voľbe iných podmienok treba dávať pozor, aby došlo k ich naplneniu.

Pseudo-kód: Model rastúcich neurónových plynov

1. Inicializácia:
   Náhodne umiestni dva neuróny a_1, a_2 vo vstupnom priestore. Množinu neurónov značíme ako A. Množina hrán je prázdna $E = \emptyset$.

2. Predloženie vzoru:
   Náhodne vyber trénovací vzor $\vec{x} \in X$ a ten predlož na vstupy neurónovej sieti.

3. Výber víťazných neurónov:
   Nájdi neurón c_1, ktorého váhový vektor je najbližší k predloženému trénovaciemu vzoru $\vec{x}$ a označ ho ako víťaza. Ďalej nájdi neurón c_2, ktorého váhový vektor je druhý najbližší k predloženému trénovaciemu vzoru $\vec{x}$.

$$c_1 = \arg\min_j \left\|\vec{w}_j - \vec{x}\right\| \quad \forall j \in A$$
$$c_2 = \arg\min_j \left\|\vec{w}_j - \vec{x}\right\| \quad \forall j \in A \setminus \{c_1\} \tag{6.15}$$

   Definujme množinu priamych susedov neurónu c_1 predpisom:

$$N_1(c_1) = \{j, d(c_1, j) = 1\} \tag{6.16}$$

4. Vytvorenie hrán:
   Ak neexistuje hrana medzi neurónmi c_1 a c_2, tak ju vytvor.

$$\text{E}= E \cup \{(c_1, c_2)\} \tag{6.17}$$

5. Priradenie lokálnej chyby $Error(c_1)$ víťaznému neurónu c_1:
   Víťaznému neurónu c_1 aktualizuj hodnotu lokálnej chyby podľa predpisu:

$$Error(c_1) = Error(c_1) + \left\|\vec{w}_{c_1} - \vec{x}\right\|^2 \tag{6.18}$$

6. Aktualizácia váh:
   Váhový vektor víťazného neurónu aktualizuj podľa predpisu:

$$\vec{w}_{c_1} = \vec{w}_{c_1} + \epsilon_b(\vec{x} - \vec{w}_{c_1}) \tag{6.19}$$

   Váhové vektory priamych susedov víťazného neurónu aktualizuj podľa predpisu:

$$\vec{w}_j = \vec{w}_j + \epsilon_n\left(\vec{x} - \vec{w}_j\right) \qquad \forall j \in N_1(c_1) \tag{6.20}$$

7. Aktualizácia veku hrán:
   Všetkým hranám vychádzajúcim z neurónu c_1 zvýš vek.

$$vek_{(C_1,j)} = vek_{(C_1,j)} + 1 \qquad \forall j \in N_1(c_1) \tag{6.21}$$

   Ak existuje hrana medzi neurónmi c_1 a c_2, tak nastav jej vek na nulu.

$$vek_{(C_1,C_2)} = 0 \tag{6.22}$$

   Ďalej odstráň všetky hrany, ktoré majú vek väčší ako $vek_{\max}$. Ak po odstránení týchto hrán vznikne neurón, z ktorého nevychádza žiadna hrana, tak tento neurón taktiež odstráň.

8. Rozhodnutie o pridaní neurónov:
   Ak počet doposiaľ vykonaných iterácií algoritmu je násobkom parametra λ, pokračuj ďalším krokom algoritmu. V opačnom prípade pokračuj na krok algoritmu číslo 2.

9. Určenie neurónov, medzi ktoré sa vloží nový neurón:
   Nájdi neurón q s najväčšou lokálnou chybou

$$q = \arg\max_{j} Error(j) \qquad \forall j \in A \tag{6.23}$$

   Z množiny najbližších susedov neurónu q nájdi neurón f s najväčšou lokálnou chybou

$$f = \arg\max_{j'} Error(j') \qquad \forall j' \in N_1(q) \tag{6.24}$$

10. Vloženie nového neurónu:
    Do stredu hrany spájajúcej neuróny q a f vlož nový neurón r.

$$A = A \cup \{r\} \tag{6.25}$$

    Vytvor váhový vektor $\vec{w}_r$ pre nový neurón r podľa predpisu:

$$\vec{w}_r = \frac{\vec{w}_q + \vec{w}_f}{2} \tag{6.26}$$

    Zmaž hranu spájajúcu neuróny q, f a vytvor dve nové hrany spájajúce neuróny q, r a r, f.

$$\begin{aligned} E &= E \cup \{(q,r),(r,f)\} \\ E &= E \setminus \{(q,f)\} \end{aligned} \tag{6.27}$$

Neurónom q a f uprav hodnotu lokálnej chyby podľa predpisu:

$$Error(q) = \alpha Error(q)$$

$$Error(f) = \alpha Error(f) \tag{6.28}$$

$$0 < \alpha < 1$$

Novému neurónu r nastav rovnakú lokálnu chybu ako má neurón q po aktualizacií svojej lokálnej chyby.

$$Error(r) = Error(q) \tag{6.29}$$

11. Aktualizácia hodnôt lokálnych chýb:
    Všetkým neurónom uprav hodnoty lokálnych chýb podľa predpisu:

$$Error(j) = \beta Error(j) \quad 0 < \beta < 1, \ \forall j \in A \tag{6.30}$$

12. Ukončenie algoritmu:
    Ak nie je dosiahnutý požadovaný počet neurónov pokračuj na krok algoritmu číslo 2.

Keďže hrany počas učiaceho procesu starnú a niektoré sa aj odstránia, dochádza vo vstupnom priestore k javu, že topologická štruktúra môže mať rôznu dimenzionalitu v rôznych oblastiach vstupného priestoru. Toto môže pomerne dosť zneprehľadniť prípadnú vizualizáciu výsledkov.

7. RBF-siete

RBF (Radial Basis Function) sieť je viacvrstvová neurónová sieť s dopredným šírením signálu. Tento model neurónových sieti sa radí k mladším modelom neurónových sieti. V osemdesiatych rokoch sa oblasť numerickej matematiky začala zaoberať interpoláciou a aproximáciou dát s využitím tzv. radiálne bázových funkcií [41], [46]. Využiť tieto funkcie k vytvoreniu nového modelu neurónových sieti navrhol Broomhead a Lowe [7] v roku 1988. K ďalšiemu rozvoju RBF sieti prispeli Moody a Darken [32].

Radiálne bázovú funkciu si môžeme predstaviť ako funkciu určenú nejakým významným bodom, ktorá pre všetky rovnako vzdialené argumenty od tohto bodu dáva rovnaké funkčné hodnoty. V dvojrozmernom priestore, pokiaľ je vzdialenosť argumentov meraná v Euklidovej metrike (2.1), tvoria množiny bodov rovnakých funkčných hodnôt kružnice. Z tohto dôvodu tieto funkcie označujeme ako radiálne.

RBF siete často dosahujú lepšie výsledky ako klasické neurónové siete, a to aj vďaka použitiu radiálne bázových funkcií, ktoré viac odpovedajú recepčným poliam skutočných neurónov.

7.1. RBF jednotka

RBF jednotka realizuje zvolenú radiálnu bázickú funkciu. Má I reálnych vstupov, na ktoré sa predkladajú vstupné vektory $\vec{x} \in R^I$. RBF jednotka si udržiava informáciu o pozícii stredu $\vec{c} = (c_1, c_2, \dots, c_I)$, pričom každý vstup RBF jednotky $x_i \in R$ má priradenú váhu c_i. Táto váha vyjadruje i-tú súradnicu stredu $\vec{c}$ odpovedajúcej RBF jednotky. Ďalej má RBF jednotka jeden reálny výstup y vyjadrujúci aktivitu jednotky. Obrázok č. 13 znázorňuje architektúru RBF jednotky.

Vnútorný potenciál ξ RBF jednotky vyjadruje vzdialenosť predloženého vzoru $\vec{x}$ od stredu $\vec{c}$ RBF jednotky. Hodnotu ξ vypočítame pomocou vzťahu:

$$\xi = \|\vec{x} - \vec{c}\| \tag{7.1}$$

Vzdialenosť vektorov $\|\vec{x} - \vec{c}\|$ sa počíta vo zvolenej metrike. V teórii RBF sieti sa používa hlavne Euklidovská metrika (2.1).

Výstupnú hodnotu y RBF jednotky spočítame ako hodnotu prenosovej funkcie φ v bode ξ.

$$y = \varphi(\xi) \tag{7.2}$$

Za prenosovú funkciu φ sa môže zvoliť napríklad niektorá z nižšie uvedených funkcií:

- Lineárna funkcia

$$\varphi(\xi) = \xi \tag{7.3}$$

- Multi-kvadratická funkcia

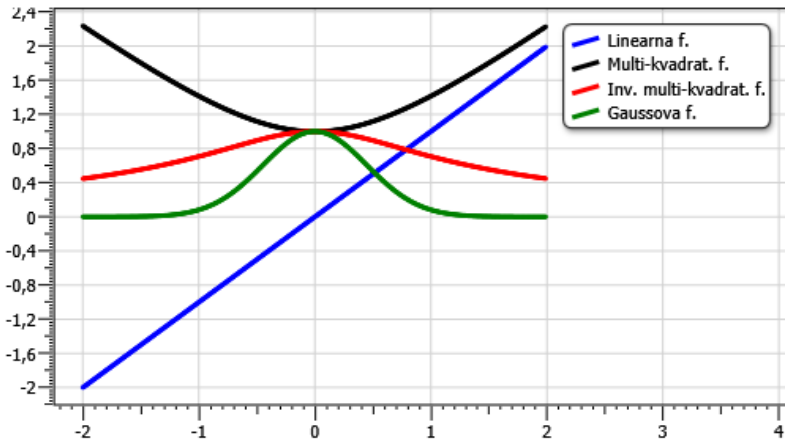
$$\varphi(\xi) = \sqrt{(\xi^2 + \sigma^2)}, \qquad \sigma > 0 \tag{7.4}$$

- Inverzná multi-kvadratická funkcia

$$\varphi(\xi) = \frac{1}{\sqrt{(\xi^2 + \sigma^2)}}, \qquad \sigma > 0 \tag{7.5}$$

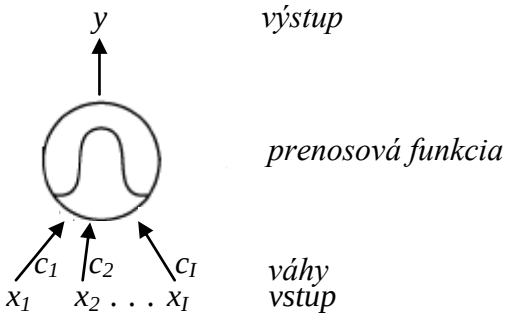
- Gaussova funkcia

$$\varphi(\xi) = \exp\left(-\frac{\xi^2}{2\sigma^2}\right), \qquad \sigma > 0 \tag{7.6}$$



Obrázok č. 12: Priebeh funkcií (lineárna, multi-kvadratická, inverzná multi-kvadratická, Gaussova funkcia), hodnota parametra šírky $\sigma = 1, r \in \langle -2, 2 \rangle$.

Najčastejšie používanou prenosovou funkciu je Gaussova funkcia, ktorá najviac zodpovedá lokálnemu charakteru RBF jednotiek. Čím bližšie k stredu RBF jednotky sa vstupný vzor nachádza, tým viac je odpovedajúca RBF jednotka aktívna.



Obrázok č. 13: Architektúra RBF jednotky

7.2. Architektúra RBF siete

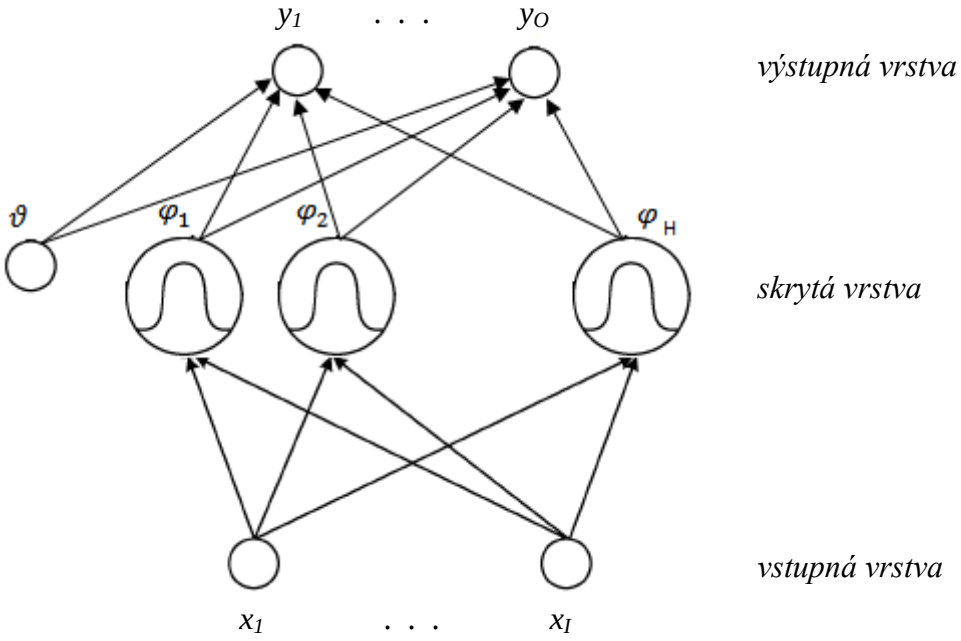
RBF sieť je trojvrstvová neurónová sieť, pričom každá vrstva plní rôzny účel. Vstupná vrstva je tvorená I neurónmi a slúži len k prenosu vstupných dát do druhej skrytej vrstvy. Skrytá vrstva je tvorená H RBF jednotkami (7.1), ktoré realizujú nelineárne zobrazenie pomocou jednotlivých radiálnych bázových funkcií. RBF jednotky sa nazývajú aj ako skryté jednotky. Výstupná vrstva je tvorená O výstupnými neurónmi s je lineárnou prenosovou funkciou.

Každý vstupný neurón je spojený s každou RBF jednotkou v skrytej vrstve, pričom spoj medzi i-tým vstupným neurónom a h-tou RBF jednotkou má priradenú váhu $c_{ih}, i = 1, \dots, I,\ h = 1, \dots, H$. Táto váha c_{ih} vyjadruje i-tú súradnicu stredu $\vec{c}_h$ h-tej RBF jednotky.

Každá RBF jednotka je spojená synaptickou váhou s každým neurónom vo výstupnej vrstve, presnejšie w_{hj} je hodnota váhy medzi h-tou RBF jednotkou a j-tým výstupným neurónom.

Do skrytej vrstvy môžeme pridať jednu špeciálnu „RBF jednotku“, ktorá bude reprezentovať prah ϑ. Táto RBF jednotka má konštantnú výstupnú hodnotu rovnú jednotke a podobne ako RBF jednotky, je spojená synaptickou váhou $\vec{w}_{H+1,j}$ s každým neurónom j vo výstupnej vrstve. Typicky sa však prah v RBF sieťach nevyužíva.

RBF sieť prevádza dve rôzne transformácie. Prvá transformácia je nelineárna zo vstupného priestoru R^I do priestoru R^H a je prevádzaná RBF jednotkami. Druhá transformácia je už lineárna z priestoru R^H do výstupného priestoru R^O a je realizovaná neurónmi vo výstupnej vrstve.



Obrázok č. 14: Architektúra RBF siete tvorenej I vstupnými neurónmi, H RBF jednotkami a O výstupnými neurónmi.

Celý proces návrhu architektúry RBF siete môžeme rozdeliť na dve základné fázy [38]. V prvej fáze ide o vývoj RBF jednotiek a v druhej fáze o optimalizáciu hodnôt váh neurónov vo výstupnej vrstve. Prvá fáza má rozhodujúci vplyv na výslednú efektivitu fungovania RBF siete a je oveľa náročnejšia ako druhá optimalizačná fáza. Je to z dôvodu, že v prvej fáze sa musia riešiť nasledujúce problémy:

- určiť typ a počet RBF jednotiek je základným problémom pri návrhu architektúry RBF siete
- určiť rozmiestnenie RBF jednotiek vo vstupnom priestore má dôležitý vplyv na funkčnosť RBF siete
- optimalizácia parametrov jednotlivých RBF jednotiek je podstatná pre správnu funkčnosť RBF siete

Problémom, ktorý sa spája s odhadom počtu RBF jednotiek, sa zaoberal i Cover vo svojej práci [8]. Vo svojej práci dospel k výsledku, že je vhodné voliť väčší počet skrytých RBF jednotiek ako je dimenzia vstupu.

7.3. Coverov teorém

Problém klasifikácie vzorov, ktorý je nelineárne transformovaný do mnohorozmerného priestoru, bude v tomto priestore lineárne separabilný pravdepodobnejšie ako v pôvodnom menej rozmernom priestore.

Nech $X = \{\vec{x}_p \in R^I, p = 1, \dots, P\}$ je množina P vstupných vzorov, pričom každý z týchto vzorov patrí do jednej z množín $X_1, X_2 \subseteq X, X_1 \cap X_2 = \emptyset, X_1 \cup X_2 = X$. Pre každý vzor $\vec{x} \in X$ definujeme vektorovú funkciu predpisom

$$\varphi(\vec{x}) = \left(\varphi_1(\vec{x}), \varphi_2(\vec{x}), \dots, \varphi_H(\vec{x})\right) \tag{7.7}$$

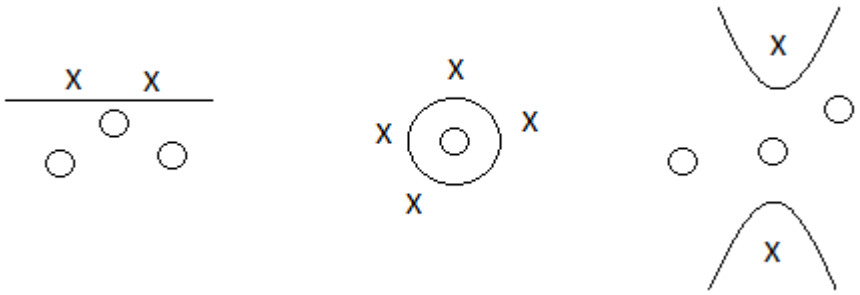
Táto funkcia $\varphi(\vec{x})$ namapuje vzor $\vec{x}$ z n-rozmerného vstupného priestoru do nového H-rozmerného priestoru. Funkcie $\varphi_h(\vec{x}), h = 1, \dots, H$ nazývame skryté funkcie, pretože ich účel je podobný skrytým jednotkám v dopredných RBF neurónových sieťach. Podobne priestor, ktorý pokrývajú tieto skryté funkcie nazývame skrytý priestor.

Dichotómiu $\{X_1, X_2\}$ množiny X nazveme φ-separabilnú, ak existuje vektor $\vec{u}$, $\vec{u} = (u_1, u_2, \dots, u_H)$ taký, že platí:

$$\begin{aligned} \vec{u}\varphi(\vec{x}) > 0, \quad & \vec{x} \in X_1 \\ \vec{u}\varphi(\vec{x}) < 0, \quad & \vec{x} \in X_2 \end{aligned} \tag{7.8}$$

Deliacu nadrovinu v skrytom priestore potom definujeme rovnosťou:

$$\vec{u}\varphi(\vec{x}) = 0 \tag{7.9}$$

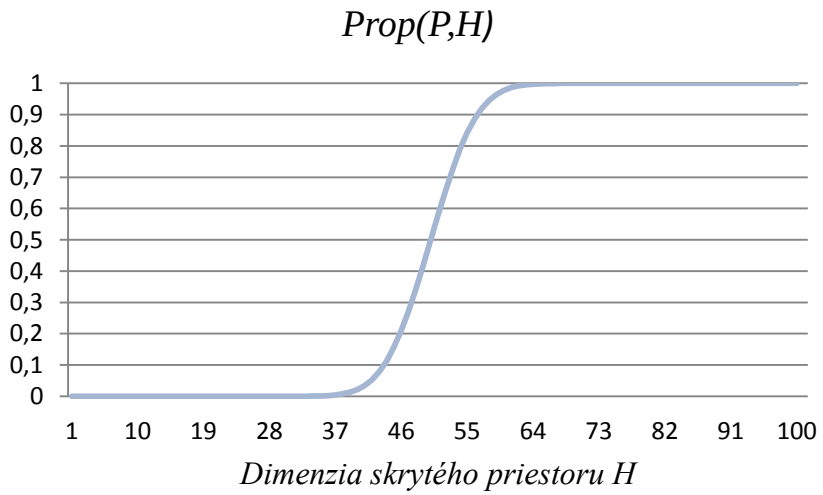


Obrázok č. 15: Príklad rôznych φ-separabilných dichotómií v 2 dvojrozmernom priestore. V poradí zľava: lineárne, sféricky a kvadraticky separabilná dichotómia [11].

Predpokladajme, že všetky možné dichotómie množiny X sú rovnako pravdepodobné. Označme $Prop(P,H)$ pravdepodobnosť, že náhodne vybraná dichotomia je φ-separabilná. Táto pravdepodobnosť je daná predpisom [8]:

$$Prop(P,H) = \left(\frac{1}{2}\right)^{P-1} \sum_{m=0}^{H-1} \binom{P-1}{m} \tag{7.10}$$

Z predchádzajúceho vzťahu vyplýva, že čím väčšia dimenzionalita H skrytého priestoru sa zvolí, tým je hodnota pravdepodobnosti $Prop(P,H)$ bližšie k jednotke.



Obrázok č. 16: Graf zobrazuje vplyv zväčšovania dimenzionality skrytého priestoru H na $Prop(P,H)$. Počet vstupných vzorov $P = 100$.

Niekedy však postačuje samotná nelinearita zobrazenia, aby sa pôvodný lineárne neseparabilný problém zmenil bez zvyšovania dimenzionality skrytého priestoru na lineárne separabilný problém. Ako príklad poslúži logická funkcia XOR [11].

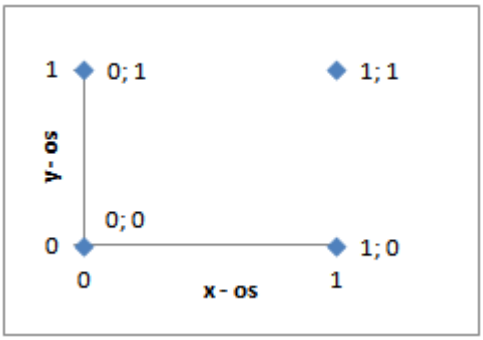
Logická funkcia XOR pre dve vstupné logické premenné A, B nadobúda nasledujúce pravdivostné ohodnotenie.

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

(7.11)

Tabuľka č. 1: Tabuľka pravdivostných hodnôt pre logickú funkciu XOR

Umiestnime body $[0,0],[0,1],[1,0],[1,1]$ do 2-rozmerného vstupného priestoru.



Obrázok č. 17: Zobrazenie vstupných vzorov v 2-rozmernom vstupnom priestore.

Vo vstupnom priestore chceme vytvoriť klasifikátor vzorov, ktorý pre body $[0,0],[1,1]$ bude dávať na výstupe hodnotu 0 a pre body $[0,1],[1,0]$ bude dávať na výstupe hodnotu 1. Definujme dve skryté funkcie φ_1, φ_2, ktoré realizujú Gaussovu funkciu so stredom umiestneným v $\vec{c}_1, \vec{c}_2$ predpisom:

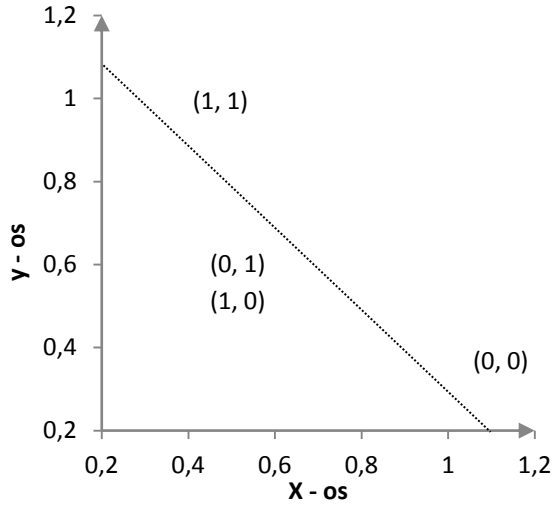
$$\varphi_1(\vec{x}) = \exp\left(-\frac{\|\vec{x}-\vec{c}_1\|^2}{2\sigma_1^2}\right), \qquad \vec{c}_1 = [1,1]^T \tag{7.12}$$

$$\varphi_2(\vec{x}) = \exp\left(-\frac{\|\vec{x}-\vec{c}_2\|^2}{2\sigma_2^2}\right), \qquad \vec{c}_2 = [0,0]^T \tag{7.13}$$

Po predložení vstupných vzorov $\vec{x} \in \{[0,0],[0,1],[1,0],[1,1]\}$ vracajú skryté funkcie $\varphi_1(\vec{x}), \varphi_2(\vec{x})$ nasledovné hodnoty:

$\vec{x}$	$\varphi_1(\vec{x})$	$\varphi_2(\vec{x})$
$(0,0)$	0,367879	1
$(0,1)$	0,606531	0,606531
$(1,0)$	0,606531	0,606531
$(1,1)$	1	0,367879

Tabuľka č. 2: Tabuľka zobrazuje funkčné hodnoty skrytých funkcií $\varphi_1(\vec{x}), \varphi_2(\vec{x})$ pre $\vec{x} \in \{[0,0],[0,1],[1,0],[1,1]\}$. Šírka oboch skrytých funkcií má hodnotu $\sigma_1 = \sigma_2 = 1$.



Obrázok č. 18: Body $[0,0]$, $[1,1]$ sú lineárne separabilné od bodov $[0,1]$, $[1,0]$.

Pre vstupné vzory platí $\vec{x} \in R^2$, a taktiež pre skryté funkcie platí $\varphi_1(\vec{x}), \varphi_2(\vec{x}) \in R^2$. Čiže aj bez nárastu dimenzionality skrytého priestoru sme previedli problém XOR na lineárne separabilný.

7.4. Učiaci proces RBF sieti

Definujme trénovaciu množinu T, ktorej prvky sú usporiadané dvojice tvaru (vstupný vzor, požadovaný výstup). Nech vektor $\vec{x} = (x_1, x_2, \dots, x_I)$ je vstupný vzor, ktorý predkladáme na vstupy neurónovej sieti. Vektor $\vec{d} = (d_1, d_2, \dots, d_O)$ je požadovaný výstup neurónovej siete na predložený vstupný vzor $\vec{x}$ a reprezentuje požadované výstupné hodnoty výstupných neurónov. Vektor $\vec{y} = (y_1, y_2, \dots, y_O)$ je skutočný výstup neurónovej siete na predložený vstupný vzor $\vec{x}$ a reprezentuje skutočné hodnoty výstupných neurónov. Trénovaciu množinu obsahujúcu P trénovacích vzorov definujeme predpisom:

$$T = \left\{ [\vec{x}_p, \vec{d}_p]; 1 \leq p \leq P,\ \vec{x}_p \in R^I, \vec{d}_p \in R^O \right\} \tag{7.14}$$

RBF jednotky typicky realizujú Gaussovú funkciu, so stredom $\vec{c}$ a šírkou σ, definovanú predpisom:

$$\varphi(\vec{x}) = \exp\left(-\frac{\|\vec{x} - \vec{c}\|^2}{2\sigma^2} \right), \tag{7.15}$$

Skutočný výstup $\vec{y}$ RBF siete spočítame pomocou vzorca:

$$y_j(\vec{x}) = \sum_{h=1}^{H} w_{hj}\, \varphi_h(\vec{x}), \qquad j = 1, \dots, O \tag{7.16}$$

$y_j(\vec{x})$ je hodnota skutočného výstupu j-tého neurónu vo výstupnej vrstve na predložený vstupný vzor $\vec{x}$. V prípade, že v RBF sieti máme aj prah ϑ, skutočný výstup $\vec{y}$ RBF siete spočítame pomocou aktualizovaného vzorca:

$$y_j(\vec{x}) = \sum_{h=1}^{H+1} w_{hj}\, \varphi_h(\vec{x}), \qquad j = 1, \dots, O \tag{7.17}$$

Pričom platí:

$$\forall \vec{x} | \varphi_{H+1}(\vec{x}) = 1 \tag{7.18}$$

Dôležitým faktom je, že rôzne vrstvy RBF siete vykonávajú rôzne úlohy. Z tohto dôvodu môžeme učiaci proces RBF siete rozdeliť na dve samostatné úlohy. Prvá úloha je spojená s rozmiestnením a nastavením parametrov jednotlivých RBF jednotiek. V druhej úlohe ide o nastavenie hodnôt synaptických váh, ktoré spájajú RBF jednotky s neurónmi vo výstupnej vrstve. Proces vývoja RBF jednotiek je oveľa pomalší oproti úlohe nastavenia synaptických váh výstupných neurónov.

Pre RBF siete bolo vyvinutých niekoľko učiacich algoritmov. V nasledujúcom texte si predstavíme algoritmus náhodnej voľby stredov RBF jednotiek (7.4.1) a hybridný učiaci algoritmus (7.5).

7.4.1. Náhodná voľba stredov RBF jednotiek

Jedná sa o pomerne jednoduchý učiaci algoritmus [11]. V prvom kroku náhodne rozmiestnime jednotlivé RBF jednotky vo vstupnom priestore, a to spôsobom, že náhodne vyberieme H rôznych vstupných vzorov $\vec{x}$ z trénovacej množiny T a tie prehlásime za stredy $\vec{c}$ jednotlivých RBF jednotiek. Keďže vstupné vzory vyberáme náhodne, aby sme zvýšili pravdepodobnosť rozmiestnenia RBF jednotiek po celom vstupnom priestore, je potreba voliť veľkú hodnotu H.

Za radiálne bázovú funkciu zvolíme Gaussovú funkciu (5.2), kde šírka σ je určená vzhľadom k rozmiestneniu stredov RBF jednotiek vo vstupnom priestore.

$$\sigma = \frac{d_{\max}}{\sqrt{2H}} \tag{7.19}$$

$d_{\max}$ vyjadruje maximálnu vzdialenosť medzi zvolenými stredmi RBF jednotiek. V oblastiach vstupného priestoru s nižšou hustotou výskytu vstupných dát môžeme hodnotu šírky σ zväčšiť.

Výstup Gaussovej funkcie so stredom umiestneným v bode $\vec{c}_h, h = 1, \dots, H$ vyjadríme predpisom:

$$\varphi(\vec{x}) = \exp\left(-\frac{H}{d_{\max}^2}\|\vec{x} - \vec{c}_h\|^2\right), \qquad h = 1, \dots, H \tag{7.20}$$

7.4.2. Pseudo-inverzná metóda

V druhom kroku je potreba nastaviť správne hodnoty synaptických váh medzi RBF jednotkami a neurónmi vo výstupnej vrstve. Toto môžeme vykonať priamo pomocou pseudo-inverznej matice [7]. Chceme riešiť nasledujúci vzťah:

$$w = G^+ d \tag{7.21}$$

d je požadovaný výstup neurónovej siete. G^+ je označenie pseudo-inverznej matice k matici $G = \{g_{ih}\}$, ktorej prvky sú definované predpisom:

$$g_{ih} = \exp\left(-\frac{2H}{d_{\max}^2}\|\vec{x}_i - \vec{c}_h\|^2\right), \qquad i = 1, \dots P, h = 1, \dots, H \tag{7.22}$$

$\vec{x}_i$ je i-tý vzor z trénovacej množiny.

Pseudo-inverznú maticu spočítame pomocou metódy SVD (Singular Value Decomposition) rozkladu.

Ak G je reálna matica rozmerov $N \times M$, potom existujú ortogonálne matice

$U = \{\vec{u}_1, \vec{u}_2, \dots, \vec{u}_N\}, V = \{\vec{v}_1, \vec{v}_2, \dots, \vec{v}_M\}$ také, že

$$U^T G V = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_K), \qquad K = \min(M, N) \tag{7.23}$$

pričom $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_K > 0$.

Čísla $\sigma_1, \sigma_2, \dots, \sigma_K$ nazývame singulárne hodnoty matice matice G. Pseudo-inverznú maticu G^+ rozmerov $M \times N$ k matici G definujeme predpisom:

$$G^+ = V \Sigma^+ U^T, \tag{7.24}$$

kde Σ^+ je matica rozmerov $N \times N$ definovaná predpisom:

$$\Sigma^+ = \text{diag}\left(\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, \ldots, \frac{1}{\sigma_K}, 0, \ldots, 0\right) \tag{7.25}$$

Zmienený postup návrhu architektúry a učenia RBF siete pomocou náhodnej voľby stredov RBF jednotiek je vhodný v prípade, že máme dostupnú rozsiahlu množinu trénovacích vzorov. V opačnom prípade nemusí RBF sieť dosahovať požadovanú výkonnosť.

7.5. Hybridný učiaci proces RBF sieti

Nevýhodou metódy náhodnej voľby stredov RBF jednotiek (7.4.1) bola požiadavka na rozsiahlu množinu trénovacích vzorov. Čo však nie je vždy možné zaručiť. Možnosťou ako sa tomuto problému vyhnúť je použiť hybridný učiaci ([11]) proces pozostávajúci s dvoch úplne odlišných štýlov učenia neurónových sieti. Jedná sa o:

1. Samo-organizačné učenie (self-organizing learning), ktorého účelom je odhadnúť vhodné rozmiestnenie stredov jednotlivých RBF jednotiek.
2. Učenie s učiteľom (supervised learning), ktorého účelom je nastaviť správne hodnoty synaptických váh medzi RBF jednotkami a neurónmi vo výstupnej vrstve.

Samo-organizačné učenie

Potrebujeme metódu, pomocou ktorej rozdelíme vstupné vzory do zhlukov. Zároveň po týchto zhlukoch požadujeme, aby boli čo možno najviac homogénne. Každý zhluk je reprezentovaný vektorom umiestneným do ťažiska zhluku, ktorý nazývame centroid. Následne do týchto centroidov príslušných zhlukov umiestnime stredy jednotlivých RBF jednotiek. Vhodným algoritmom, ktorý spĺňa tieto požiadavky, je napríklad K-means klastrovací algoritmus (0). Tento algoritmus rozdelí vstupný priestor do vopred určeného počtu klastrov. Výsledné klastre K-means algoritmu sú kompaktné a navzájom disjunktné. Vhodné je ale aj použitie Kohonenových máp (5).

Za radiálne bázovú funkciu zvolíme Gaussovu funkciu (5.2), kde šírka σ je jednoznačne určená vzhľadom k rozmiestneniu stredov RBF jednotiek vo vstupnom priestore.

$$\sigma = \frac{d_{\min}}{2} \tag{7.26}$$

$d_{\min}$ vyjadruje vzdialenosť k stredu najbližšej RBF jednotky v Euklidovej metrike.

Učenie s učiteľom

V tejto časti algoritmu chceme nastaviť príslušné hodnoty synaptických váh medzi RBF jednotkami a neurónmi vo výstupnej vrstve pomocou metódy najmenších štvorcov LMS (least-mean-square). LMS algoritmus patrí medzi gradientné algoritmy a vyznačuje sa predovšetkým svojou rýchlosťou.

Nech $T = \{[\vec{x}_p, \vec{d}_p]; 1 \leq p \leq P,\ \vec{x}_p \in R^I, \vec{d}_p \in R^O\}$ je množina P trénovacích vzorov. Pre každý predložený vzor $\vec{x}_p$ na vstup RBF sieti dáva skrytá vrstva RBF jednotiek výstup $\varphi(\vec{x}_p) = \left(\varphi_1(\vec{x}_p), \varphi_2(\vec{x}_p), \dots, \varphi_H(\vec{x}_p)\right)$, kde H je počet RBF jednotiek. Na vstup LMS algoritmu budeme predkladať práve tieto vektory, ktoré vzniknú ako výstup skrytej vrstvy tvorenej RBF jednotkami. Vytvoríme novú pozmenenú trénovaciu množinu $T' = \{[\varphi(\vec{x}_p), \vec{d}_p]; 1 \leq p \leq P, \varphi(\vec{x}_p) \in R^H, \vec{d}_p \in R^O\}$, ktorej prvky budeme predkladať na vstup LMS algoritmu. $\eta(t)$ určuje hodnotu parametra učenia a platí $0 \leq \eta(t) \leq 1$.

7.5.1. Least-mean-square algoritmus

Pseudo-kód:

1. Inicializácia:
   Zvoľ malé náhodné hodnoty synaptických váh $w_{hj}, h = 1, \dots, H, j = 1, \dots, O$, pričom H určuje počet skrytých RBF jednotiek, O určuje počet neurónov vo výstupnej vrstve.
2. Predloženie trénovacieho vzoru:
   Predlož na vstup nový trénovací vzor tvaru $[\varphi(\vec{x}_p), \vec{d}_p]$ ([vstup, požadovaný výstup]).

3. Výpočet skutočného výstupu:
   Pre zadaný vstupný vzor $\varphi(\vec{x}_p)$ spočítame hodnotu reálneho výstupu RBF siete $\vec{y}_p$ podľa:

$$y_{pj} = \sum_{h=0}^{H} w_{hj} \varphi_h(\vec{x}_p), \qquad j = 1, \dots, O \tag{7.27}$$

   y_{pj} je skutočný výstup j-tého výstupného neurónu na predložený vstupný vzor $\varphi(\vec{x}_p)$.

4. Výpočet chybovej hodnoty:
   Spočítaj odchýlku siete $\vec{e}_p$ medzi požadovaným výstupom $\vec{d}_p$ a skutočným výstupom $\vec{y}_p$ siete na predložený vstupný vzor $\varphi(\vec{x}_p)$ podľa predpisu:

$$\vec{e}_p = \vec{d}_p - \vec{y}_p \tag{7.28}$$

5. Aktualizácia váh:
   Aktualizuj synaptické váhy podľa predpisu:

$$w_{hj} = w_{hj} + \eta\varphi(\vec{x}_p)\vec{e}_p, \quad p = 1, \dots, P, h = 1, \dots, H, j = 1, \dots, O \tag{7.29}$$

6. Ukončenie algoritmu:
   Pokračuj na krok algoritmu číslo 2 a predlož sieti nový trénovací vzor.

Použitie Kohonenových máp (5), príp. K-means algoritmu (0), k identifikácii oblastí vo vstupnom priestore so zvýšeným výskytom vstupných vzorov, do ktorých sa následne umiestnia jednotlivé RBF jednotky sa javí výhodne. Týmto postupom získame dobré pokrytie vstupného priestoru RBF jednotkami, avšak môžu vzniknúť aj redundantné RBF jednotky.

Algoritmus náhodnej voľby RBF jednotiek (7.4.1) náhodne vyberá vstupné vzory a tie sa prehlásia za stredy RBF jednotiek. Aby sa zvýšila pravdepodobnosť pokrytia celého vstupného priestoru RBF jednotkami, musí byť veľké množstvo vstupných vzorov prehlásených za RBF jednotky. Vznikajú tak RBF siete, ktoré obsahujú zbytočne veľké množstvo redundantných RBF jednotiek. Z tohto dôvodu sa javí hybridný učiaci proces výhodnejší.

Problém redundantných RBF jednotiek sa snažia riešiť algoritmy (8), ktoré umožňujú aktualizovať topológiu RBF sieti počas jej učenia.

8. RBF siete s adaptatívnou topológiou

Pre RBF siete vzniklo viacero učiacich algoritmov, ktoré umožňujú počas procesu učenia siete meniť topológiu RBF siete. Tieto algoritmy môžeme rozlišovať podľa dvoch faktorov:

- Aký druh aktualizačných operácií s topológiou RBF siete podporujú
- V akej fáze učiaceho algoritmu sa aktualizačné operácie vykonávajú

Učiace algoritmy RBF sieti, ktoré v sebe nemajú implementovanú operáciu odstraňovania redundantných RBF jednotiek, často produkujú RBF siete zbytočne veľkých rozmerov. V prípade, že algoritmus podporuje operáciu odstraňovania RBF jednotiek, je dôležité, aby sa kontrolovalo splnenie kritéria pre odstránenie RBF jednotiek počas celého priebehu učiaceho algoritmu. Ak k odstráneniu nadbytočných RBF jednotiek zo siete dochádza až po predložení všetkých vstupných vzorov, tak RBF sieť počas učiaceho algoritmu nie je kompaktná [44], [42]. V mnohých reálnych aplikáciách týchto učiacich algoritmov totižto nie je možné počas ich priebehu určiť, či už boli predložené všetky vstupné vzory. Z tohto dôvodu býva ťažké rozhodnúť, v ktorom okamihu učiaceho algoritmu sa má vykonať fáza odstránenia RBF jednotiek.

Algoritmy, ktoré podporujú iba operáciu pridania novej RBF jednotky do skrytej vrstvy RBF siete fungujú v princípe nasledovne. Na začiatku učiaceho procesu začíname s prázdnou množinou RBF jednotiek. Postupne sa do siete pridávajú nové RBF jednotky, ktoré najviac prispievajú k znižovaniu celkovej chyby RBF siete. Medzi takéto algoritmy patria napr. Metóda dopredného výberu (Forward selection) [37], [36], RAN(8.1), RANEKF (8.1).

Učiace algoritmy, ktoré podporujú iba operáciu odstránenia RBF jednotky zo skrytej vrstvy RBF siete fungujú na nasledovnom princípe. Na začiatku učiaceho procesu je každému vstupnému vzoru priradená jedna RBF jednotka. Postupne sa odstraňujú také RBF jednotky, ktoré najmenej prispievajú k znižovaniu celkovej chyby RBF siete. Medzi algoritmy tohto typu patri napr. Metóda spätnej eliminácie (Backward elimination) [12],[13].

Poslednú skupinu tvoria učiace algoritmy, ktoré podporujú operáciu pridania i odstránenia RBF jednotky do/zo skrytej vrstvy RBF siete. Tieto algoritmy fungujú na

nasledovnom princípe. Na začiatku učiaceho procesu začíname s prázdnou množinou RBF jednotiek. Počas učenia sa do siete postupne pridávajú nové RBF jednotky. V prípade, že v danom kroku algoritmu nedošlo k pridaniu novej RBF jednotky, prevedie sa kontrola RBF jednotiek. V závislosti na konkrétnom učiacom algoritme sa testujú na prípadne odstránenie zo skrytej vrstvy, buď všetky RBF jednotky alebo len RBF jednotka, ktorá je najbližšie aktuálne predloženému vstupnému vzoru. Medzi tieto algoritmy patria napr. MRAN (8.2) – testujú sa všetky RBF jednotky na prípadne odstránenie, EMRAN (8.3), GAP_RBF (8.4).

8.1. Resource allocating network via Extended Kalman filter (RANEKF)

V roku 1991 J.C.Platt [39] navrhol pre RBF neurónové siete novy učiaci algoritmus Resource allocating network (RAN). Algoritmus RAN podporuje pridávanie nových RBF jednotiek do skrytej vrstvy RBF siete. Odstraňovanie redundantných RBF jednotiek však už nepodporuje. Tento algoritmus začína s prázdnou množinou RBF jednotiek. Postupne sa do siete pridávajú nové RBF jednotky, ktoré najviac prispievajú k znižovaniu celkovej chyby RBF siete. K nastaveniu hodnôt synaptických váh medzi jednotlivými RBF jednotkami a výstupnými neurónmi sa používa metóda najmenších štvorcov (7.5.1).

V. Kadirkamanathan a M. Niranjan navrhli vo svojej práci [20] použiť k nastaveniu parametrov RBF jednotiek (stred, šírka) a hodnôt synaptických váh rozšírený Kalmanov filter - EKF (8.5) namiesto metódy najmenších štvorcov. Vznikol tak nový algoritmus RANEKF.

Algoritmus RANEKF

Nech $T = \{[\vec{x}_p, \vec{d}_p]; 1 \leq p \leq P,\ \vec{x}_p \in R^I, \vec{d}_p \in R^O\}$ je množina P trénovacích vzorov. Parameter $e_{\min}$ udáva hodnotu požadovanej aproximačnej presnosti siete. Parameter ϵ_i vyjadruje minimálnu vzdialenosť predloženého vzoru $\vec{x}$ od stredu $\vec{c}_{nr}$ najbližšej RBF jednotky, aby sa mohla vytvoriť nová RBF jednotka, ktorej stred bude totožný s predloženým vzorom $\vec{x}$. Vektor $\vec{e}_p$ vyjadruje odchýlku siete medzi požadovaným výstupom $\vec{d}_p$ a skutočným výstupom $\vec{y}_p$ siete na predložený vstupný vzor $\vec{x}_p$. Parameter κ určuje mieru prekrytia odpovedi RBF jednotiek vo vstupnom priestore.

Na začiatku učiaceho procesu RBF sieť neobsahuje žiadnu skrytú RBF jednotku. Postupne budeme predkladať na vstup sieti jednotlivé trénovacie vzory. Pre každý predložený vzor $\vec{x}_p$ spočítame celkový výstup RBF siete podľa vzťahu:

$$y_j = \sum_{h=1}^{H} w_{hj} \exp\left(-\frac{\left\|\vec{x}_p - \vec{c}_h\right\|^2}{2\sigma_h^2}\right), \qquad j = 1, \ldots, O \tag{8.1}$$

Spočítame hodnoty parametrov ϵ_p a $\vec{e}_p$, ktoré sú potrebné pri rozhodovaní o pridaní novej RBF jednotky podľa nasledujúcich vzťahov:

$$\epsilon_p = \max\{\epsilon_{\max}\gamma^p, \epsilon_{\min}\},\ 0<\gamma<1 \tag{8.2}$$

$$\vec{e}_p = \vec{d}_p - \vec{y}_p \tag{8.3}$$

Konštanta γ má zoslabujúci účinok na parameter $\epsilon_{\max}$. Hranice intervalu $\epsilon_{\min}$, $\epsilon_{\max}$ pre výpočet hodnoty ϵ_p zadáva užívateľ. Ak je posledný predložený vzor $\vec{x}_p$ dostatočne vzdialený od stredu $\vec{c}_{nr}$ najbližšej RBF jednotky a zároveň významnosť eventuálne pridanej RBF jednotky bude väčšia ako požadovaná minimálna aproximačná presnosť siete $e_{\min}$, tak sa vytvorí nová RBF jednotka. Parametre novo vzniknutej RBF jednotky nastavíme podľa vzťahov:

$$\begin{aligned} \vec{w}_{H+1} &= \vec{e}_p \\ \vec{c}_{H+1} &= \vec{x}_p \\ \sigma_{H+1} &= \kappa\left\|\vec{x}_p - \vec{c}_{nr}\right\| \end{aligned} \tag{8.4}$$

Ak neboli splnené podmienky pre pridanie novej RBF jednotky, tak aktualizujeme parametre $\vec{w}_h, \vec{c}_h, \sigma_h, h \in H$ všetkých RBF jednotiek pomocou algoritmu EKF (8.5).

Pseudo-kód: RANEKF algoritmus

1. Výpočet výstupu RBF siete:
   Vypočítaj celkový výstup RBF siete podľa predpisu:

$$y_j = \sum_{h=1}^{H} w_{hj} \exp\left(-\frac{\left\|\vec{x}_p - \vec{c}_h\right\|^2}{2\sigma_h^2}\right), \qquad j = 1, \ldots, O \tag{8.5}$$

   kde O určuje počet výstupných neurónov a H určuje počet skrytých RBF jednotiek.

2. Výpočet hodnôt rastových kritérií:
   Vypočítaj hodnoty parametrov určujúcich hodnoty kritérií, ktoré sú potrebné pri rozhodovaní o zvýšení počtu RBF jednotiek podľa:

$$\epsilon_p = \max\{\epsilon_{\max}\gamma^p, \epsilon_{\min}\},\ 0<\gamma<1 \tag{8.6}$$

$$\vec{e}_p = \vec{d}_p - \vec{y}_p \tag{8.7}$$

3. Rozhodnutie o pridaní novej RBF jednotky:
   Vyhodnoť nasledujúce kritéria:

$$\left\|\vec{x}_p - \vec{c}_{nr}\right\| > \epsilon_p \tag{8.8}$$

$$\left\|\vec{e}_p\right\| > e_{\min} \tag{8.9}$$

   kde $\vec{x}_p$ je posledný predložený vzor na vstup RBF sieti.

   - Ak boli splnené obe kritéria pre zvýšenie počtu RBF jednotiek, tak vytvor novú RBF jednotku a nastav jej parametre podľa nasledujúcich vzťahov:

$$\vec{w}_{H+1} = \vec{e}_p$$

$$\vec{c}_{H+1} = \vec{x}_p \tag{8.10}$$

$$\sigma_{H+1} = \kappa\left\|\vec{x}_p - \vec{c}_{nr}\right\|$$

     $\vec{w}_{H+1}$ je vektor vyjadrujúci hodnoty synaptických váh vychádzajúcich z novo vzniknutej $H + 1$ RBF jednotky.

   - Ak nie sú splnené obe kritéria pre zvýšenie počtu RBF jednotiek, tak uprav hodnoty parametrov $\vec{w}_h, \vec{c}_h, \sigma_h, h \in H$ všetkých RBF jednotiek pomocou algoritmu EKF (8.5).

4. Ukončenie algoritmu:
   Pokiaľ sú na vstupe ďalšie trénovacie vzory, pokračuj na krok algoritmu číslo 1.

V 3. kroku RANEKF algoritmu, ak nedôjde k vytvoreniu novej RBF jednotky do siete, je vždy potrebné aktualizovať parametre všetkých RBF jednotiek. Keďže v algoritme EKF (8.5) sa vykonáva množstvo maticových operácii, s narastajúcim počtom RBF jednotiek výrazne rastie časová zložitosť algoritmu. Z tohto dôvodu je výhodné namiesto aktualizácie parametrov všetkých RBF jednotiek využiť rovnaký postup ako v algoritme EMRAN (8.3). Algoritmus EMRAN aktualizuje pomocou EKF algoritmu iba parametre najbližšej RBF jednotky (v Euklidovej vzdialenosti) k posledne predloženému vstupnému vzoru. Po zavedení tejto zmeny dochádza k výraznému zjednodušeniu a zrýchleniu výpočtov EKF algoritmu (8.3.1).

8.2. Minimal resource allocating network (MRAN)

Algoritmy RAN a RANEKF (8.1) nepodporujú operáciu odstraňovania RBF jednotiek z RBF siete. Z tohto dôvodu môže výsledná RBF sieť obsahovať množstvo redundantných RBF jednotiek. Algoritmus MRAN [52], [28] na rozdiel od algoritmov RAN, RANEKF umožňuje odstrániť RBF jednotky, ktoré majú relatívne malý vplyv na celkový výstup RBF siete.

K identifikácií redundantných RBF jednotiek sa využíva princíp posuvného okna. Veľkosť posuvného okna nám určuje počet naposledy predložených vstupných vzorov, na ktorých sa posudzuje, aký vplyv mal výstup skúmanej RBF jednotky na celkový výstup RBF siete. Rovnaký princíp sa využíva aj k určeniu RBF jednotiek, ktoré sa pridajú do skrytej vrstvy RBF siete. Voľba vhodnej veľkosti posuvného okna je veľmi závislá na rozdelení vstupných dát.

Algoritmus MRAN

Nech $T = \{[\vec{x}_p, \vec{d}_p]; 1 \leq p \leq P,\ \vec{x}_p \in R^I, \vec{d}_p \in R^O\}$ je množina P trénovacích vzorov. Parameter $e_{\min}$ udáva hodnotu požadovanej aproximačnej presnosti siete. Parameter ϵ_i vyjadruje minimálnu vzdialenosť predloženého vzoru $\vec{x}$ od stredu $\vec{c}_{nr}$ najbližšej RBF jednotky, aby sa mohla vytvoriť nová RBF jednotka, ktorej stred bude totožný s predloženým vzorom $\vec{x}$. Vektor $\vec{e}_p$ vyjadruje odchýlku siete medzi požadovaným výstupom $\vec{d}_p$ a skutočným výstupom $\vec{y}_p$ siete na predložený vstupný vzor $\vec{x}_p$. Parameter κ určuje mieru prekrytia odpovedi RBF jednotiek vo vstupnom priestore. N_{add} určuje veľkosť posuvného okna pri rozhodovaní o pridaní novej RBF jednotky. N_{del} určuje veľkosť posuvného okna pri rozhodovaní o odstránení RBF jednotky z RBF siete.

Na začiatku učiaceho procesu RBF sieť neobsahuje žiadnu skrytú RBF jednotku. Postupne budeme predkladať na vstup sieti jednotlivé trénovacie vzory. Pre každý predložený vzor $\vec{x}_p$ spočítame celkový výstup RBF siete podľa vzťahu:

$$y_j = \sum_{h=1}^{H} w_{hj} \exp\left(-\frac{\left\|\vec{x}_p - \vec{c}_h\right\|^2}{2\sigma_h^2}\right), \qquad j = 1, \ldots, O \tag{8.11}$$

Spočítame hodnoty parametrov $\epsilon_p, \vec{e}_p$ a e_{rmse}, ktoré sú potrebné pri rozhodovaní o pridaní novej RBF jednotky podľa nasledujúcich vzťahov:

$$\epsilon_p = \max\{\epsilon_{\max}\gamma^p, \epsilon_{\min}\},\ 0<\gamma<1 \tag{8.12}$$

$$\vec{e}_p = \vec{d}_p - \vec{y}_p \tag{8.13}$$

$$e_{rmse} = \sqrt{\sum_{k=p-(N_{add}-1)}^{p} \frac{e_k^2}{N_{add}}} \tag{8.14}$$

Konštanta γ má zoslabujúci účinok na parameter $\epsilon_{\max}$. Hranice intervalu $\epsilon_{\min}$, $\epsilon_{\max}$ pre výpočet hodnoty ϵ_p zadáva užívateľ. N_{add} je počet vstupných vzorov naposledy predložených na vstup RBF siete.

Ak sú splnené všetky tri nasledujúce podmienky, tak sa vytvorí nová RBF jednotka.

$$\left\|\vec{x}_p - \vec{c}_{nr}\right\| > \epsilon_p \tag{8.15}$$

$$\vec{e}_p > e_{\min} \tag{8.16}$$

$$e_{rmse} > e'_{\min} \tag{8.17}$$

Parameter $e'_{\min}$ udáva minimálnu hodnotu pre strednú kvadratickú odchýlku e_{rmse} na posledných N_{add} predložených vzorov. Hodnotu parametra $e'_{\min}$ zadáva užívateľ.

V prípade vytvorenia novej RBF jednotky nastavíme parametre novej RBF jednotky podľa nasledujúcich vzťahov:

$$\begin{aligned} \vec{w}_{H+1} &= \vec{e}_p \\ \vec{c}_{H+1} &= \vec{x}_p \\ \sigma_{H+1} &= \kappa\left\|\vec{x}_p - \vec{c}_{nr}\right\| \end{aligned} \tag{8.18}$$

Ak neboli splnené všetky podmienky pre pridanie novej RBF jednotky, tak aktualizujeme parametre $\vec{w}_h, \vec{c}_h, \sigma_h, h \in H$ všetkých RBF jednotiek pomocou algoritmu EKF (8.5). Po aktualizácií parametrov RBF jednotiek, nasleduje kontrola všetkých RBF jednotiek na prípadne odstránenie z RBF siete.

Spočítame normalizovaný výstup $y'_{hj}(\vec{x}_k)$ na j-tom výstupnom neuróne z h-tej RBF jednotky na predložený vzor $\vec{x}_k$ podľa predpisu:

$$y'_{hj}(\vec{x}_k) = \frac{y_{hj}(\vec{x}_k)}{\max\{y_{1j}(\vec{x}_k), y_{2j}(\vec{x}_k), \dots, y_{Hj}(\vec{x}_k)\}}$$
$$h = 1, \dots H, j = 1, \dots, O, k = \big(p - (N_{del} - 1)\big), \dots, p \tag{8.19}$$

Hodnotu výstupu $y_{hj}(\vec{x}_k)$ na j-tom výstupnom neuróne z h-tej RBF jednotky na predložený vzor $\vec{x}_k$ spočítame podľa predpisu:

$$y_{hj}(\vec{x}_k) = w_{hj}\exp\left(-\frac{\|\vec{x}_k - \vec{c}_h\|^2}{2\sigma_h^2}\right),$$
$$h = 1, \dots H, j = 1, \dots, O, k = \big(p - (N_{del} - 1)\big), \dots, p \tag{8.20}$$

kde w_{hj} je hodnota synaptickej váhy z h-tej RBF jednotky do j-teho výstupného neurónu, H je počet RBF jednotiek, O je počet výstupných neurónov.

RBF jednotku odstránime, ak normalizovaný výstup $y'_{hj}(\vec{x}_k)$, $j = 1, \dots, O$ h-tej RBF jednotky k posledným N_{del} predloženým vzorom $\vec{x}_k, k = \big(p - (N_{del} - 1)\big), \dots, p$ má hodnotu na každom výstupnom neuróne menšiu ako je zvolená hodnota parametra δ.

$$y'_{hj}(\vec{x}_k) < \delta,$$
$$h = 1, \dots H, j = 1, \dots, O, k = \big(p - (N_{del} - 1)\big), \dots, p \tag{8.21}$$

p je index posledného predloženého vzoru na vstup RBF sieti, parameter δ zadáva užívateľ.

Zásadným problémom algoritmu MRAN je jeho časová zložitosť. V prípade, že nedôjde k pridaniu novej RBF jednotky do RBF siete, aktualizujú sa vždy parametre všetkých RBF jednotiek. S narastajúcim počtom RBF jednotiek, sa výrazne zhoršuje časová zložitosť algoritmu EKF (8.5), ktorý sa používa k aktualizácii parametrov RBF jednotiek. Vylepšením algoritmu MRAN, vznikol algoritmus EMRAN (8.3), ktorý tento problém prekonal. Algoritmus EMRAN aktualizuje pomocou EKF algoritmu iba parametre najbližšej RBF jednotky (v Euklidovej vzdialenosti) k posledne predloženému vstupnému vzoru.

Pseudo-kód: MRAN algoritmus

1. Výpočet výstupu RBF siete:
   Vypočítaj celkový výstup RBF siete podľa predpisu:

$$y_j = \sum_{h=1}^{H} w_{hj} \exp\left(-\frac{\left\|\vec{x}_p - \vec{c}_h\right\|^2}{2\sigma_h^2}\right), \qquad j = 1, \ldots, O \tag{8.22}$$

   kde O určuje počet výstupných neurónov a H určuje počet skrytých RBF jednotiek.

2. Výpočet hodnôt rastových kritérií:
   Vypočítaj hodnoty parametrov určujúcich hodnoty kritérií, ktoré sú potrebné pri rozhodovaní o zvýšení počtu RBF jednotiek podľa:

$$\epsilon_p = \max\{\epsilon_{\max}\gamma^p, \epsilon_{\min}\}, \ 0<\gamma<1 \tag{8.23}$$

$$\vec{e}_p = \vec{d}_p - \vec{y}_p \tag{8.24}$$

$$e_{rmse} = \sqrt{\sum_{k=p-(N_{add}-1)}^{p} \frac{e_k^2}{N_{add}}} \tag{8.25}$$

3. Rozhodnutie o pridaní novej RBF jednotky:
   Vyhodnoť nasledujúce kritéria:

$$\left\|\vec{x}_p - \vec{c}_{nr}\right\| > \epsilon_p \tag{8.26}$$

$$\vec{e}_p > e_{\min} \tag{8.27}$$

$$e_{rmse} > e'_{\min} \tag{8.28}$$

   kde $\vec{x}_p$ je posledný predložený vzor na vstup RBF sieti.

   - Ak boli splnené všetky uvedené kritéria pre zvýšenie počtu RBF jednotiek, tak vytvor novú RBF jednotku a nastav jej parametre podľa nasledujúcich vzťahov:

$$\vec{w}_{H+1} = \vec{e}_p$$

$$\vec{c}_{H+1} = \vec{x}_p \tag{8.29}$$

$$\sigma_{H+1} = \kappa\left\|\vec{x}_p - \vec{c}_{nr}\right\|$$

$\vec{w}_{H+1}$ je vektor vyjadrujúci hodnoty synaptických váh vychádzajúcich z novo vzniknutej $H+1$ RBF jednotky. Pokračuj na krok algoritmu č. 6.

- Ak nie sú splnené všetky kritéria pre pridanie novej RBF jednotky, tak uprav hodnoty parametrov $\vec{w}_h, \vec{c}_h, \sigma_h, h \in H$ všetkých RBF jednotiek pomocou algoritmu EKF (8.5). Pokračuj na krok algoritmu č. 4.

4. Výpočet normalizovaného výstupu RBF jednotiek
   K posledným N_{del} predloženým vzorom spočítaj normalizovaný výstup $y'_{hj}(\vec{x}_k)$ pre každú RBF jednotku podľa predpisu:

$$y'_{hj}(\vec{x}_k) = \frac{y_{hj}(\vec{x}_k)}{\max\{y_{1j}(\vec{x}_k), y_{2j}(\vec{x}_k), \dots, y_{Hj}(\vec{x}_k)\}} \tag{8.30}$$
$$h = 1, \dots H, j = 1, \dots, O, k = \big(p - (N_{del} - 1)\big), \dots, p$$

   $y_{hj}(\vec{x}_k)$ je hodnota výstupu na j-tom výstupnom neuróne z h-tej RBF jednotky na predložený vzor $\vec{x}_k$:

$$y_{hj}(\vec{x}_k) = w_{hj}\exp\left(-\frac{\|\vec{x}_k - \vec{c}_h\|^2}{2\sigma_h^2}\right), \tag{8.31}$$
$$h = 1, \dots H, j = 1, \dots, O, k = \big(p - (N_{del} - 1)\big), \dots, p$$

   kde w_{hj} je hodnota synaptickej váhy z h-tej RBF jednotky do j-teho výstupného neurónu, H je počet RBF jednotiek, O je počet výstupných neurónov.

5. Rozhodnutie o odstránení redundantných RBF jednotiek:
   Odstráň každú h-tú RBF jednotku, ktorej normalizovaný výstup $y'_{hj}(\vec{x}_k)$, $j = 1, \dots, O$ k posledným N_{del} predloženým vzorom $\vec{x}_k, k = \big(p - (N_{del} - 1)\big), \dots, p$ má hodnotu na každom výstupnom neuróne menšiu ako zvolená hodnota parametra δ.

$$y'_{hj}(\vec{x}_k) < \delta, \tag{8.32}$$
$$h = 1, \dots H, j = 1, \dots, O, k = \big(p - (N_{del} - 1)\big), \dots, p$$

   p je index posledného predloženého vzoru na vstup RBF sieti

6. Ukončenie algoritmu:
   Pokiaľ sú na vstupe ďalšie trénovacie vzory, pokračuj na krok algoritmu číslo 1.

8.3. Extended minimal resource allocating network (EMRAN)

Algoritmus MRAN (8.2) naráža na problém časovej náročnosti. V prípade, že po predložení nového vzoru RBF sieti nedôjde k pridaniu novej RBF jednotky, aktualizujú sa parametre (stred, šírka, synaptické váhy) všetkých RBF jednotiek. K aktualizácií parametrov RBF jednotiek sa používa algoritmus EKF (8.5), ktorého časová zložitosť s narastajúcim počtom RBF jednotiek výrazne rastie. Vylepšením algoritmu MRAN vzniká nový algoritmus EMRAN [28], ktorý tento problém prekonáva.

Algoritmus EMRAN implementuje princíp „víťaz berie všetko“ (“winner takes all”). Pod víťaznou RBF jednotkou rozumejme najbližšiu RBF jednotku (v Euklidovej vzdialenosti) k poslednému predloženému vstupnému vzoru. Algoritmus EMRAN na rozdiel od algoritmu MRAN (8.2) v prípade, že nedôjde k pridaniu novej RBF jednotky do RBF siete, aktualizuje pomocou algoritmu EKF (8.5) iba parametre víťaznej RBF jednotky. Toto je jediný ale zásadný rozdiel medzi algoritmom MRAN a EMRAN. Vplyv tejto zmeny na časovú náročnosť algoritmu EKF je nasledujúci.

8.3.1. Výpočtová zložitosť algoritmu EKF v algoritmoch MRAN, EMRAN

Algoritmus EKF (8.5) počas svojho behu pracuje s maticou Kalmanovho úžitku $K_{(Z\times O)}$, gradientnou maticou $B_{(Z\times O)}$, maticou šumu $R_{(O\times O)}$ a kovariančnou maticou $P_{(Z\times Z)}$. Z určuje počet všetkých parametrov, ktoré bude algoritmus EKF aktualizovať.

$$Z = H \times (O + I + 1), \tag{8.33}$$

kde O je dimenzia výstupného priestoru RBF siete, I je dimenzia vstupného priestoru RBF siete a H je počet RBF jednotiek v skrytej vrstve RBF siete.

Ak sa zameriame iba na počet maticových násobení, ktoré sa vykonajú v i-tom behu EKF algoritmu, dostávame nasledujúce údaje:

- k výpočtu súčinu matíc $K_i B_i^T P_{i-1}$ je potrebné vykonať $Z^3 + Z^2 \times O$ násobení
- k získaniu Kalmanovej matice úžitku $K_{i_{(Z\times O)}}$ je potreba vykonať $Z^2 \times O + Z \times O^2$ násobení
- k výpočtu súčinu matíc $K_i e_i$ je potrebné vykonať $Z \times O$ násobení

Celkovo je teda potrebné vykonať $Z^3 + 2 \times Z^2 \times O + Z \times (O^2 + O)$ násobení, čím sa dostávame na polynomiálnu rovnicu tretieho stupňa, pričom hodnota Z má zásadný vplyv na časovú zložitosť EKF algoritmu. S narastajúcim počtom RBF jednotiek sa hodnota Z zväčšuje a tým narastá časová zložitosť algoritmu.

8.4. Basic growing and pruning RBF (GAP-RBF)

Algoritmus GAP [15], [14] je sekvenčný učiaci algoritmus pre RBF neurónové siete. Podobne ako algoritmy MRAN (8.2) a EMRAN (8.3) aj algoritmus GAP umožňuje z RBF siete odstránenie RBF jednotiek, ktoré majú relatívne malý vplyv na celkový výstup RBF siete. Proces identifikácie a následného odstránenia redundantnej RBF jednotky sa však v algoritme GAP oproti algoritmom MRAN a EMRAN výrazne zrýchlil. Algoritmus GAP totižto neprevádza test na prípadne odstránenie na všetkých RBF jednotkách ale vždy testuje iba jednu RBF jednotku. Konkrétne RBF jednotku, ktorá je najbližšie (v Euklidovej vzdialenosti) k poslednému predloženému vstupnému vzoru. Dôvod je nasledujúci.

Uvažujme RBF sieť, ktorá neobsahuje redundantné RBF jednotky. Predložíme tejto sieti na vstup nový vstupný vzor, ktorý nevyvolá vznik novej RBF jednotky. K tomuto predloženému vzoru sa nájde najbližšia RBF jednotka (v Euklidovej vzdialenosti). Pomocou algoritmu EKF (8.5) sa aktualizujú parametre iba tejto RBF jednotky, čiže parametre ostaných RBF jednotiek zostali nezmenené. Keďže RBF sieť neobsahovala redundantné RBF jednotky, tak jedine aktualizovaná RBF jednotka môže stratiť svoju významnosť. Z tohto dôvodu stačí vždy skontrolovať len najbližšiu RBF jednotku k predloženému vzoru a jedine táto RBF jednotka môže byť v danom okamihu odstránená. Tento fakt, že v každom kroku algoritmu pracujeme s parametrami len jednej RBF jednotky, výrazne zrýchľuje učiaci algoritmus.

Algoritmus GAP-RBF

Nech $T = \{[\vec{x}_p, \vec{d}_p]; 1 \leq p \leq P,\ \vec{x}_p \in R^I, \vec{d}_p \in R^O\}$ je množina P trénovacích vzorov. Parameter $e_{\min}$ udáva hodnotu požadovanej aproximačnej presnosti siete. Parameter ϵ_i vyjadruje minimálnu vzdialenosť predloženého vzoru $\vec{x}$ od stredu $\vec{c}_{nr}$ najbližšej RBF jednotky, aby sa mohla vytvoriť nová RBF jednotka, ktorej stred bude totožný s predloženým vzorom $\vec{x}$. Vektor $\vec{e}_p$ vyjadruje odchýlku siete medzi požadovaným výstupom $\vec{d}_p$ a skutočným výstupom $\vec{y}_p$ siete na predložený vstupný vzor $\vec{x}_p$. Parameter κ určuje mieru prekrytia odpovedi RBF jednotiek vo vstupnom priestore.

Na začiatku učiaceho procesu RBF sieť neobsahuje žiadnu skrytú RBF jednotku. Postupne budeme predkladať na vstup sieti jednotlivé trénovacie vzory. Pre každý predložený vzor $\vec{x}_p$ spočítame celkový výstup RBF siete podľa vzťahu:

$$y_j = \sum_{h=1}^{H} w_{hj} \exp\left(-\frac{\left\|\vec{x}_p - \vec{c}_h\right\|^2}{2\sigma_h^2}\right), \qquad j = 1, \ldots, O \tag{8.34}$$

Spočítame hodnoty parametrov $\epsilon_p, \vec{e}_p$, ktoré sú potrebné pri rozhodovaní o pridaní novej RBF jednotky podľa nasledujúcich vzťahov:

$$\epsilon_p = \max\{\epsilon_{\max}\gamma^p, \epsilon_{\min}\}, \ 0<\gamma<1 \tag{8.35}$$

$$\vec{e}_p = \vec{d}_p - \vec{y}_p \tag{8.36}$$

Konštanta γ má zoslabujúci účinok na parameter $\epsilon_{\max}$. Hranice intervalu $\epsilon_{\min}$, $\epsilon_{\max}$ pre výpočet hodnoty ϵ_p zadáva užívateľ.

Ak sú splnené všetky tri nasledujúce podmienky, tak sa vytvorí nová RBF jednotka.

$$\left\|\vec{x}_p - \vec{c}_{nr}\right\| > \epsilon_p \tag{8.37}$$

$$\vec{e}_p > e_{\min} \tag{8.38}$$

$$\frac{\left\|\vec{e}_p\right\|\left(1.8\kappa\left\|\vec{x}_p - \vec{c}_{nr}\right\|\right)^p}{S(X)} > e_{\min} \tag{8.39}$$

V prípade vytvorenia novej RBF jednotky nastavíme parametre novej RBF jednotky podľa nasledujúcich vzťahov:

$$\begin{aligned} \vec{w}_{H+1} &= \vec{e}_p \\ \vec{c}_{H+1} &= \vec{x}_p \\ \sigma_{H+1} &= \kappa\left\|\vec{x}_p - \vec{c}_{nr}\right\| \end{aligned} \tag{8.40}$$

Ak neboli splnené podmienky pre pridanie novej RBF jednotky, tak upravíme parametre $\vec{w}_{nr}, \vec{c}_{nr}, \sigma_{nr}$ najbližšej RBF jednotky pomocou EKF algoritmu (8.5). Následne skontrolujeme túto upravenú RBF jednotku, či jej vplyv na celkový výstup neklesol pod požadovanú hranicu $e_{\min}$. Ak je splnená nasledujúca podmienka, odstránime RBF jednotku zo siete.

$$\left\|\frac{\vec{w}_{nr}(1.8\sigma_{nr})^p}{S(X)}\right\| < e_{\min} \tag{8.41}$$

$S(X)$ je odhad rozsahu intervalu, z ktorého sú vyberané vstupné vzory. V prípade , že vstupné dáta transformujeme do intervalu $\langle 0, 1\rangle$ je hodnota $S(X) = 1$. Pokračujeme predložením ďalšieho trénovacieho vzoru na vstup RBF sieti.

Pseudo-kód: GAP-RBF algoritmus

1. Výpočet výstupu RBF siete:
   Vypočítaj celkový výstup RBF siete podľa predpisu:

$$y_j = \sum_{h=1}^{H} w_{hj} \exp\left(-\frac{\left\|\vec{x}_p - \vec{c}_h\right\|^2}{2\sigma_h^2}\right), \qquad j = 1, \ldots, O \tag{8.42}$$

   kde O určuje počet výstupných neurónov a H určuje počet skrytých RBF jednotiek.

2. Výpočet hodnôt rastových kritérií:
   Vypočítaj hodnoty parametrov určujúcich hodnoty kritérií, ktoré sú potrebné pri rozhodovaní o zvýšení počtu RBF jednotiek podľa:

$$\epsilon_p = \max\{\epsilon_{\max}\gamma^p, \epsilon_{\min}\}, \ 0<\gamma<1 \tag{8.43}$$

$$\vec{e}_p = \vec{d}_p - \vec{y}_p \tag{8.44}$$

3. Rozhodnutie o pridaní novej RBF jednotky:
   Vyhodnoť nasledujúce kritéria:

$$\left\|\vec{x}_p - \vec{c}_{nr}\right\| > \epsilon_p \tag{8.45}$$

$$\vec{e}_p > e_{\min} \tag{8.46}$$

$$\frac{\left\|\vec{e}_p\right\|\left(1.8\kappa\left\|\vec{x}_p - \vec{c}_{nr}\right\|\right)^p}{S(X)} > e_{\min} \tag{8.47}$$

   kde $\vec{x}_p$ je posledný predložený vzor na vstup RBF sieti.

   - Ak boli splnené všetky uvedené kritéria pre zvýšenie počtu RBF jednotiek, tak vytvor novú RBF jednotku a nastav jej parametre podľa nasledujúcich vzťahov:

$$\vec{w}_{H+1} = \vec{e}_p$$

$$\vec{c}_{H+1} = \vec{x}_p \tag{8.48}$$

$$\sigma_{H+1} = \kappa\left\|\vec{x}_p - \vec{c}_{nr}\right\|$$

     $\vec{w}_{H+1}$ je vektor vyjadrujúci hodnoty synaptických váh vychádzajúcich z novo vzniknutej $H+1$ RBF jednotky. Pokračuj na krok algoritmu č. 6.

- Ak nie sú splnené všetky kritéria pre pridanie novej RBF jednotky, tak upravíme parametre $\vec{w}_{nr}, \vec{c}_{nr}, \sigma_{nr}$ najbližšej RBF jednotky pomocou EKF algoritmu (8.5). Ak vplyv aktualizovanej RBF jednotky na celkový výstup klesol pod požadovanú hranicu $e_{\min}$, odstráň RBF jednotku s indexom nr.

$$\left\| \frac{\vec{w}_{nr}(1.8\sigma_{nr})^p}{S(X)} \right\| < e_{\min} \tag{8.49}$$

4. Ukončenie algoritmu:
   Pokiaľ sú na vstupe ďalšie trénovacie vzory, pokračuj na krok algoritmu číslo 1.

8.4.1. Recursive growing and pruning RBF (RGAP)

Základná verzia GAP algoritmu predkladá trénovacie vzory na vstup RBF siete iba jeden krát. To je príčinou, že RBF sieť nie je po ukončení procesu učenia dostatočne naučená, je tzv. „pod trénovaná“. Odstránenie tohto nedostatku prináša nový učiaci algoritmus pre RBF siete RGAP [15]. Algoritmus RGAP funguje na nasledovnom princípe. Ak je na vstup RBF siete predložený nový trénovací vzoru RGAP funguje ako základná verzia GAP (8.4). V okamihu, keď na vstupe RBF siete nie je predložený žiadny trénovací vzor, začnú sa na vstup opätovne predkladať už predložené trénovacie vzory.

8.5. Extended Kalman Filter

Kalmanov filter sa využíva pre odhad budúcich stavov, do ktorých sa môže skúmaný model dostať. Rozšírený Kalmanov filter EKF [52] sa využíva v prípadoch, keď skúmaný model nie je možné popísať lineárnymi rovnicami. V našom prípade využijeme EKF k predikcii jednotlivých parametrov RBF jednotiek. Parametre Kalmanovho filtra sa v každom kroku aktualizujú na základe aktuálne dostupných informácii o stave skúmaného modelu. Nový filter teda vzniká úpravou predchádzajúceho filtra, ktorá spracúva novo získane informácie.

Algoritmus EKF

Nech $\vec{m}$ je vektor, ktorého zložky tvoria parametre jednotlivých RBF jednotiek.

$$\vec{m} = [\vec{w}_0^T, \vec{w}_1^T, \vec{c}_1^T, \sigma_1, \dots, \vec{w}_H^T, \vec{c}_H^T, \sigma_H] \tag{8.50}$$

Vektor $\vec{m}_i$ získame aktualizáciou z predchádzajúceho vektora $\vec{m}_{i-1}$ nasledovne:

$$\vec{m}_p = \vec{m}_{p-1} + K_p e_p \tag{8.51}$$

$$Z = |\vec{m}_p| = H \times (O + I + 1) + O, \tag{8.52}$$

$$\vec{e}_p = \vec{d}_p - \vec{y}_p \tag{8.53}$$

$\vec{e}_p$ vyjadruje odchýlku siete medzi požadovaným výstupom $\vec{d}_p$ a skutočným výstupom $\vec{y}_p$ siete na predložený vstupný vzor $\vec{x}_p$. K_p je Kalmanov úžitkový vektor definovaný predpisom:

$$K_{p_{(Z\times O)}} = S_{p-1_{(Z\times Z)}} B_{p_{(Z\times O)}} \left[R_{p_{(O\times O)}} + B_{p\,(Z\times O)}^T S_{p-1_{(Z\times Z)}} B_{p_{(Z\times O)}} \right]^{-1} \tag{8.54}$$

$B_p = \nabla_{\vec{m}} f(\vec{x}_p)$ je gradientná matica funkcie $f(\vec{x}_p)$ s ohľadom na vektor parametrov $\vec{m}$ definovaná predpisom:

$$B_{p_{(Z\times O)}} = \tag{8.55}$$

$$[I_{(O\times O)}, \varphi_1(\vec{x}_p) I_{(O\times O)}, \varphi_1(\vec{x}_p) \frac{2\vec{w}_1}{\sigma_1^2} (\vec{x}_p - \vec{c}_1)^T, \varphi_1(\vec{x}_p) \frac{2\vec{w}_1}{\sigma_1^3} \|\vec{x}_p - \vec{c}_1\|^2, \dots,$$

$$\varphi_H(\vec{x}_p) I_{(O\times O)}, \varphi_H(\vec{x}_p) \frac{2\vec{w}_H}{\sigma_H^2} (\vec{x}_p - \vec{c}_H)^T, \varphi_H(\vec{x}_p) \frac{2\vec{w}_H}{\sigma_H^3} \|\vec{x}_p - \vec{c}_H\|^2]^T$$

$I_{(O\times O)}$ je jednotková matica.

Matica R_i zabezpečuje istú mieru šumu a je definovaná predpisom:

$$R_{p_{(O\times O)}} = \begin{pmatrix} r_0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & r_0 \end{pmatrix} \tag{8.56}$$

S_p je kovariančná matica chýb odhadu, ktorá sa inicializuje nasledovne:

$$S_{p_{(Z\times Z)}} = \begin{pmatrix} s_0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & s_0 \end{pmatrix} \tag{8.57}$$

Matica S_p je pozitívne definitná symetrická matica a v priebehu EKF algoritmu sa aktualizuje podľa vzorca:

$$S_p = \left[I_{(Z\times Z)} - K_p B_p^T\right] S_{p-1} + q_0 I_{(Z\times Z)} \tag{8.58}$$

$I_{(Z\times Z)}$ je jednotková matica. q_0 je skalár, ktorý určuje prípustné náhodné kroky v smere gradientu vektora.

Po zväčšení počtu RBF jednotiek, sa zväčší dimenzia S_p nasledovne:

$$S_p = \begin{pmatrix} S_{p-1} & 0 \\ 0 & s_0 I_{(Z_1\times Z_1)} \end{pmatrix} \tag{8.59}$$

Z_1 je rozmer jednotkovej matice a vyjadruje počet nových parametrov, ktoré so sebou priniesla nová RBF jednotka.

Pseudo-kód: Extended Kalman Filter

(zjednodušená verzia algoritmu EKF, aktualizujú sa parametre iba jednej RBF jednotky)

1. Inicializácia:
   Inicializuj matice R, S:

$$R_{(O\times O)} = \begin{pmatrix} r_0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & r_0 \end{pmatrix} \tag{8.60}$$

$$P_{(Z\times Z)} = \begin{pmatrix} p_0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & p_0 \end{pmatrix} \tag{8.61}$$

   $Z = |\vec{m}_p| = O + I + 1$ (dimenzia výstupných vektorov O + dimenzia vstupných vzorov I + dimenzia šírky RBF jednotky 1)

2. Vytvor vstupný vektor $\vec{m}_p$:
   Vektor $\vec{m}_p$ tvoria parametre RBF jednotky, ktorej stred $\vec{c}$ je najbližšie k predloženému trénovaciemu vzoru $\vec{x}_p$.

$$nr = \arg\min_h \left(\left\|\vec{x}_p - \vec{c}_h\right\|\right), \qquad \forall h \in H \tag{8.62}$$

$$\vec{m}_p = [\vec{w}_{nr}^T, \vec{c}_{nr}^T, \sigma_{nr}] \tag{8.63}$$

3. Vytvor gradientnú maticu B_p:
   Matica B_p je definovaná predpisom:

$$B_{p_{(Z\times O)}} = \tag{8.64}$$

$$\left[\varphi_{nr}(\vec{x}_p) I_{(O\times O)}, \varphi_{nr}(\vec{x}_p) \frac{2\vec{w}_{nr}}{\sigma_{nr}^2} \left(\vec{x}_p - \vec{c}_{nr}\right)^T, \varphi_{nr}(\vec{x}_p) \frac{2\vec{w}_{nr}}{\sigma_{nr}^3} \left\|\vec{x}_p - \vec{c}_{nr}\right\|^2\right]^T$$

   $I_{(O\times O)}$ je jednotková matica.

4. Spočítaj Kalmanov úžitkový vektor podľa vzorca:

$$K_{p_{(Z\times O)}} = \tag{8.65}$$

$$S_{p-1_{(Z\times Z)}} B_{p_{(Z\times O)}} \left[R_{p_{(O\times O)}} + B_{p\,(Z\times O)}^T S_{p-1_{(Z\times Z)}} B_{p_{(Z\times O)}}\right]^{-1}$$

5. Aktualizuj vektor $\vec{m}_p$ podľa vzorca:

$$\vec{m}_p = \vec{m}_{p-1} + K_p e_p \tag{8.66}$$

6. Vytvor novú kovariančnú maticu S_p podľa vzorca:

$$S_p = \left[I_{(Z\times Z)} - K_p B_p^T\right] S_{p-1} + q_0 I_{(Z\times Z)} \tag{8.67}$$

   $I_{(Z\times Z)}$ je jednotková matica.

7. Ukončenie algoritmu:
   EKF algoritmus vráti aktualizovaný vektor $\vec{m}_p$. Parametre RBF jednotky, ktorej sa táto aktualizácia týkala, sa zmenia na parametre aktuálne obsiahnuté vo vektore $\vec{m}_p$.

8.6. Zhrnutie učiacich algoritmov pre RBF siete

Pre základný model RBF sieti sme si predstavili dva algoritmy, a to algoritmus náhodnej voľby RBF jednotiek (7.4.1) a hybridný učiaci proces (7.5). U oboch algoritmov je potrebné už vopred určiť požadovaný počet RBF jednotiek. Pri spracovaní dát, u ktorých nemáme žiadnu informáciu o ich prípadnej vnútornej štruktúre je táto požiadavka výrazne obmedzujúca. Algoritmus náhodnej voľby RBF jednotiek náhodne vyberá vzory z trénovacej množiny, ktoré následne prehlási za stredy RBF jednotiek. Predpokladom k dosiahnutiu požadovanej presnosti RBF siete je použitie veľkého počtu RBF jednotiek, čo vedie k vzniku redundantných RBF jednotiek.

Hybridný učiaci proces využíva k rozmiestneniu RBF jednotiek Kohonenové mapy, vhodný je však vďaka svojej rýchlosti aj algoritmus K-means. Toto využitie pomocnej metódy znižuje počet redundantných RBF jednotiek, pretože RBF jednotiek sú umiestňované do tých oblasti vstupného priestoru, kde je zvýšený predpoklad výskytu vstupných vzorov.

RBF siete s adaptívnou topológiou umožňujú efektívne spracovať úplne neznáme dáta. Výsledná topológia RBF siete sa postupne vyvíja počas priebehu učiaceho algoritmu. Algoritmy RAN (8.1) a RANEKF (8.1) umožňujú v prípade potreby vznik novej RBF jednotky. Odstránenie RBF jednotiek však nepodporujú. Nová RBF jednotka sa vytvorí iba v prípade, ak sa v okolí predkladaných vzorov aktuálne nevyskytuje iná RBF jednotka a zároveň významnosť (v zmysle zníženia celkovej chyby RBF siete) novej RBF jednotky dosiahne požadované hodnoty. Ak RBF jednotka v priebehu učenia stratí svoju významnosť, nie je možné ju z RBF siete odstrániť. Algoritmus RANKEF požíva k aktualizácií parametrov (stred, šírka, synaptické váhy) RBF jednotiek algoritmus EKF (8.5) na rozdiel od algoritmu RAN, ktorý využíva algoritmus LMS (7.5.1). Plat [20] vo svojej práci preukázal, že použitie algoritmu EKF k aktualizácií parametrov RBF jednotiek vedie k získaniu presnejších RBF sieti ako pri použití algoritmu LMS. Algoritmy RAN aj RANEKF v každej iterácií algoritmu aktualizujú parametre všetkých RBF jednotiek, čo vedie k zvýšením časovým nárokom.

Algoritmus MRAN (8.2) na rozdiel od algoritmov RAN, RANEKF umožňuje odstrániť RBF jednotky, ktoré majú relatívne malý vplyv na celkový výstup RBF siete. Výsledná RBF sieť by tak mala obsahovať nižší počet RBF jednotiek. K identifikácií redundantných RBF jednotiek sa využíva princíp posuvného okna. Tu však nastáva problém, pretože voľba vhodnej veľkosti posuvného okna je veľmi závislá na rozdelení vstupných dát. Zároveň je potrebné predpokladať náhodný výber trénovacích vzorov. Ak by totižto boli RBF sieti postupne predkladané vzory z jednotlivých klastrov, musela by byť veľkosť posuvného okna pre odstraňovanie RBF jednotiek veľmi veľká, takmer rovná počtu všetkých predkladaných vzorov. V opačnom prípade by sme postupne

mazali RBF jednotky objavené na začiatku algoritmu. K aktualizácií parametrov RBF jednotiek sa používa algoritmus EKF (8.5), pričom v každej iterácií algoritmu sú aktualizované parametre všetkých RBF jednotiek. Následne sa vo fáze odstraňovania RBF jednotiek testujú všetky RBF jednotky na prípadne odstránenie, čo opäť vedie k zvýšeným časovým nárokom algoritmu.

Algoritmus EMRAN (8.3) prináša zrýchlenie algoritmu MRAN. Toto zrýchlenie nastáva pri aktualizácií parametrov RBF jednotiek. V prípade, že nedôjde k pridaniu novej RBF jednotky do RBF siete, aktualizujú sa pomocou algoritmu EKF iba parametre víťaznej RBF jednotky. Pod víťaznou RBF jednotkou rozumieme najbližšiu RBF jednotku (v Euklidovej vzdialenosti) k poslednému predloženému vstupnému vzoru. U oboch algoritmov MRAN aj EMRAN pred samotným spustením natrafíme na problém spojený s iniciálnym nastavením hodnôt parametrov. Celkovo je potreba zadať hodnoty pre 13 parametrov.

Algoritmus GAP (8.4) rovnako ako algoritmy MRAN (8.2) a EMRAN (8.3) umožňuje nielen pridať, ale aj odstrániť RBF jednotky, ktoré majú relatívne malý vplyv na celkový výstup RBF siete. Proces identifikácie a následného odstránenia redundantnej RBF jednotky sa však v algoritme GAP oproti algoritmom MRAN a EMRAN výrazne zrýchlil. Algoritmus GAP vždy testuje na prípadne odstránenie iba jednu RBF jednotku. Konkrétne RBF jednotku, ktorá je najbližšie (v Euklidovej vzdialenosti) k poslednému predloženému vstupnému vzoru. GAP algoritmus vo svojej základnej verzii predkladá trénovacie vzory na vstup RBF sieti iba jeden krát, čo môže spôsobovať tzv. „pod trénovanie“ RBF siete. Odstránenie tohto nedostatku prináša upravená verzia algoritmu GAP algoritmus RGAP (8.4.1), ktorý umožňuje RBF sieti na vstup opakovane predkladať už predložené trénovacie vzory. Zvýšenie počtu trénovacích dát má opäť vplyv na nárast časovej zložitosti algoritmu.

8.7. Citlivostná analýza

Citlivostná analýza [45] je ďalšou metódou ako efektívnejšie postupovať pri návrhu RBF sieti. Ešte pred samotnou konštrukciou RBF siete prevedieme citlivostnú analýzu a na základe získaných výsledkov z tejto analýzy určíme rozmiestnenie jednotlivých RBF jednotiek vo vstupnom priestore. Citlivostná analýza pomáha určiť citlivosť celkového výstupu siete na zmenu jednotlivých parametrov v sieti. V prípade RBF sieti budeme pod pojmom citlivosť rozumieť matematický odhad druhej mocniny odchýlok siete spôsobených zmenou stredov RBF jednotiek.

Za radiálne bázické funkcie zvolíme Gaussovú funkciu definovanú predpisom:

$$\varphi(\vec{x}) = \exp\left(-\frac{\|\vec{x}-\vec{c}\|^2}{2\sigma^2}\right) \tag{8.68}$$

Reálny výstup $\vec{y} = (y_1, y_2, \dots, y_O)$ RBF siete spočítame podľa predpisu:

$$y_j = \sum_{h=1}^{H} w_{hj}\,\varphi_h(\vec{x}), \qquad j = 1,..,O \tag{8.69}$$

w_{hj} je synaptická váha medzi h-tou RBF jednotkou a j-tým výstupným neurónom.

Nech $\overrightarrow{c'}_h, \sigma'_h, \overrightarrow{w'}_h$ označujú stred, šírku a synaptické váhy h-tej perturbovanej RBF jednotky, pričom platí:

$$\overrightarrow{c'}_h = \vec{c}_h + \Delta\vec{c}_h \tag{8.70}$$

$$\overrightarrow{w'}_h = \vec{w}_h + \Delta\vec{w}_h \tag{8.71}$$

Vektor $\vec{v} = \left(\varphi_1(\vec{x}), \varphi_2(\vec{x}), \dots, \varphi_H(\vec{x})\right)$ reprezentuje výstup RBF jednotiek skrytej vrstvy na predložený vstupný vzor $\vec{x}$. Výslednú odchýlku Δy_j siete spôsobenú perturbaciou určíme:

$$\Delta y_j = \overrightarrow{w'}_j\overrightarrow{v'} - \vec{w}_j\vec{v} =$$

$$= \sum_{h=1}^{H} \overrightarrow{w'}_{hj} \exp\left(-\frac{\left\|\vec{x}-\overrightarrow{c'}_h\right\|^2}{2\sigma'^2_h}\right) - \sum_{h=1}^{H} w_{hj} \exp\left(-\frac{\|\vec{x}-\vec{c}_h\|^2}{2\sigma_h^2}\right), \tag{8.72}$$

$$j = 1, \dots, O$$

Šírky σ_h RBF jednotiek sa môžu nastaviť na základe nejakých predchádzajúcich znalostí riešeného problému**,** prípadne vzhľadom k rozmiestneniu RBF jednotiek vo vstupnom priestore podľa vzťahu:

$$\sigma_h = \frac{d_{\max}}{\sqrt{2H}}, \qquad h = 1, \dots, H \tag{8.73}$$

$d_{\max}$ vyjadruje maximálnu vzdialenosť medzi zvolenými stredmi RBF jednotiek.

Perturbaciu parametrov h-tej RBF jednotky, čiže odchylku stredu $\Delta\vec{c}_h$ a synaptických váh $\Delta\vec{w}_j$ spojených s j-tým výstupným neurónom môžeme vyjadriť pomocou Gaussovho rozdelenia s odchýlkou $\sigma_{\vec{c}_h}$ a $\sigma_{\vec{w}_j}$:

$$p(\Delta\vec{c}_h) = \frac{1}{\left(\sqrt{2\pi\sigma_{\vec{c}_h}}\right)^I} \exp\left(\frac{\Delta\vec{c}_h^T \Delta\vec{c}_h}{2\sigma_{\vec{c}_h}^2}\right)$$
$$p(\Delta\vec{w}_j) = \frac{1}{\left(\sqrt{2\pi\sigma_{\vec{w}_j}}\right)^H} \exp\left(\frac{\Delta\vec{w}_j^T \Delta\vec{w}_j}{2\sigma_{\vec{w}_j}^2}\right) \tag{8.74}$$

I je dimenzia trénovacích vzorov $\vec{x}$.

Definujme rekurzívne citlivosť [45]. Nech máme $H-1$ pevne zvolených stredov RBF jednotiek a chceme pridať stred ďalšej RBF jednotky. Citlivosť S_j^H j-tého výstupného neurónu k aktuálne H RBF stredom definujeme ako $\left(\Delta y_j\right)^2$ (štvorec odchýlok siete spôsobených perturbaciou RBF stredov) z ohľadom na všetky $\Delta\vec{c}_h$ a trénovaciu množinu T. Citlivosť S_j^H môžeme definovať predpisom:

$$S_j^H = E\left[\left(\Delta y_j\right)^2\right] = \int p(\Delta\vec{w})\, p(\Delta\vec{c}) \times \tag{8.75}$$
$$\times \sum_{\vec{x}_i \in T} \sum_{m,n=1}^{H} \overrightarrow{w'}_{mj} \overrightarrow{w'}_{nj} \exp\left(-\frac{\|\vec{x}_i - \vec{c}_m - \Delta\vec{c}_m\|^2}{2\sigma_m^2} - \frac{\|\vec{x}_i - \vec{c}_n - \Delta\vec{c}_n\|^2}{2\sigma_n^2}\right) d\Delta\vec{c}\, d\Delta\vec{w}$$
$$-2 \int p(\Delta\vec{w})\, p(\Delta\vec{c}) \times$$

$$\times \sum_{\vec{x}_i \in T} \sum_{m,n=1}^{H} \overrightarrow{w'}_{mj} w_{nj} \exp\left(-\frac{\|\vec{x}_i - \vec{c}_m - \Delta\vec{c}_m\|^2}{2\sigma_m^2} - \frac{\|\vec{x}_i - \vec{c}_n - \Delta\vec{c}_n\|^2}{2\sigma_n^2} \right) d\Delta\vec{c} d\Delta\vec{w}$$

$$+ \int p(\Delta\vec{w})\, p(\Delta\vec{c}) \times$$

$$\times \sum_{\vec{x}_i \in T} \sum_{m,n=1}^{H} w_{mj} w_{nj} \exp\left(-\frac{\|\vec{x}_i - \vec{c}_m - \Delta\vec{c}_m\|^2}{2\sigma_m^2} - \frac{\|\vec{x}_i - \vec{c}_n - \Delta\vec{c}_n\|^2}{2\sigma_n^2} \right) d\Delta\vec{c} d\Delta\vec{w}$$

$$= G_1 - 2G_2 + G_3$$

$G_1,\ G_2,\ G_3$ po integrovaní podľa $\Delta\vec{c}$ a $\Delta\vec{w}$ vyjadríme:

$$G_1 = \sum_{\vec{x}_i \in T} \sum_{m,n=1;m\neq n}^{H} w_{mj} w_{nj} \frac{\left(\sqrt{\sigma_m^2 \sigma_n^2}\right)^I}{\left(\sqrt{\left(\sigma_m^2 + \sigma_{\vec{c}_m}^2\right)\left(\sigma_n^2 + \sigma_{\vec{c}_n}^2\right)}\right)^I} \tag{8.76}$$

$$\times \exp\left(-\frac{\|\vec{x}_i - \vec{c}_m\|^2}{2\left(\sigma_m^2 + \sigma_{\vec{c}_m}^2\right)} - \frac{\|\vec{x}_i - \vec{c}_n\|^2}{2\left(\sigma_n^2 + \sigma_{\vec{c}_n}^2\right)} \right)$$

$$+ \sum_{\vec{x}_i \in T} \sum_{m=1}^{H} \left(w_{mj}^2 + \sigma_{\vec{w}_j}^2\right) \frac{\left(\sqrt{\sigma_m^2}\right)^I}{\left(\sqrt{\left(\sigma_m^2 + \sigma_{\vec{c}_m}^2\right)}\right)^I} \times \exp\left(-\frac{\|\vec{x}_i - \vec{c}_m\|^2}{\sigma_m^2 + \sigma_{\vec{c}_m}^2} \right)$$

$$G_2 = \tag{8.77}$$

$$\sum_{\vec{x}_i \in T} \sum_{m,n=1}^{H} w_{mj} w_{nj} \frac{\left(\sqrt{\sigma_m^2}\right)^I}{\left(\sqrt{\left(\sigma_m^2 + \sigma_{\vec{c}_m}^2\right)}\right)^I} \times \exp\left(-\frac{\|\vec{x}_i - \vec{c}_m\|^2}{2\left(\sigma_m^2 + \sigma_{\vec{c}_m}^2\right)} - \frac{\|\vec{x}_i - \vec{c}_n\|^2}{2\sigma_n^2} \right)$$

$$G_3 = \tag{8.78}$$

$$\sum_{\vec{x}_i \in T} \sum_{m,n=1}^{H} w_{mj} w_{nj} \exp\left(-\frac{\|\vec{x}_i - \vec{c}_m\|^2}{2\sigma_m^2} - \frac{\|\vec{x}_i - \vec{c}_n\|^2}{2\sigma_n^2} \right)$$

Najviac citlivé vektory môžeme určiť pomocou vzťahu (8.78). Stredy RBF jednotiek nemôžu byť určené len na základe citlivosti, pretože niektoré z kritických vektorov môžu byť korelované. K odstráneniu tohto problému použijeme metódu ortogonálnych najmenších štvorcov OLS.

8.7.1. Ortogonálne najmenšie štvorce

Metóda ortogonálnych najmenších štvorcov OLS (Orthogonal Least Squares) [17], [45]. Nech matica $Y = (\vec{y}_1, \vec{y}_2, \dots, \vec{y}_P)^T$ obsahuje výstupy RBF siete pre každý predložený trénovací vzor $\vec{x} \in T$, pričom platí nasledujúca rovnica v maticovom zápise:

$$Y = VW \tag{8.79}$$

Y, V, W sú matice rozmerov $Y_{P\times O}, V_{P\times P}, W_{P\times O}$.

Každú maticu dokážeme faktorizovať na súčin dvoch matíc, takže predchádzajúci vzťah môžeme prepísať nasledovne:

$$Y = (QA)W \tag{8.80}$$

Matica V je zapísaná v tvare súčinu matíc $Q_{P\times P}$ a $A_{P\times P}$. Matica Q je ortogonálna matica a matica A je horná trojuholníková matica.

$$G = \begin{pmatrix} v_{11} & v_{12} & \cdots & v_{1P} \\ v_{12} & v_{12} & & \vdots \\ \vdots & & \ddots & \vdots \\ v_{1P} & \dots & \cdots & v_{PP} \end{pmatrix} \tag{8.81}$$

$$A = \begin{pmatrix} 1 & a_{12} & \dots & & a_{1P} \\ 0 & 1 & & a_{23} & \vdots \\ & & \ddots & \ddots & \vdots \\ & & & 1 & a_{(P-1)P} \\ 0 & \dots & & 0 & 1 \end{pmatrix} \tag{8.82}$$

Maticu V chceme previesť na ortogonálny tvar, čiže všetky vektory budú navzájom na seba kolmé. V každom kroku sa ortogonalizuje jeden stĺpec matice V.

Ortogonalizačný proces popisujú nasledujúce operácie.

1. Prvý vektor $\vec{q}_1$ sa zvolí priamo:

$$\vec{q}_1 = \vec{v}_1 \tag{8.83}$$

2. Pre $p = 2, \dots, P$ opakuj nasledujúce operácie

$$a_{ip} = \frac{\vec{q}_i^T \vec{v}_p}{\|\vec{q}_i\|}, \qquad 1 \le i < p \tag{8.84}$$

$$\vec{q}_p = \vec{v}_p - \sum_{i=1}^{p-1} a_{ip}\vec{q}_i \tag{8.85}$$

8.8. Algoritmus citlivostnej analýzy

Pseudo-kód

1. Inicializácia:
   Spočítaj výsledné hodnoty RBF funkcií pre každý trénovací vzor $\vec{x}_p \in T$. Získané hodnoty dosaď za jednotlivé prvky matice V zo vzťahu $Y = VW$.

$$V = \left\{ v_{ph} | 0 < p \le P, 0 < h \le H, \vec{v}_p = \left(\varphi_1(\vec{x}_p), \varphi_2(\vec{x}_p), \dots, \varphi_H(\vec{x}_p) \right) \right\} \tag{8.86}$$

2. Voľba prvého kritického vektoru:
   Spočítaj hodnotu citlivosti pre každý stĺpec matice V pomocou vzťahu (9.8).
   - Stĺpec s najväčšou hodnotou citlivosti zvolíme sa za prvý stĺpec matice $Q^{(1)}$.
   - Spočítaj odchýlku siete $Error^{(1)}$ pre zvolený RBF stred.
   - Nastav $H = 2$.

3. Ortogonalizácia stĺpcov matice V:
   Preveď proces ortogonalizácie metódou OLS (8.7.1) na všetkých stĺpcoch matice V s využitím stĺpcov matice $Q^{(H-1)}$.

4. Výpočet:
   Pre každý trénovací vzor uvažuj $\vec{c}_p$ ako kandidáta na H-ty stred RBF jednotky, ktorý odpovedá ortogonalnému stĺpcu $\vec{q}_p$, ($H \le p \le P$).
   - Spočítaj hodnoty synaptických váh pomocou pseudo-inverznej metódy (7.4.2).

- Spočítaj hodnoty citlivosti $S^{(H)}(\vec{c}_p)$ predchádzajúcich $H-1$ RBF stredov. Hodnoty synaptických váh aktualizuj podľa vzťahu:

$$w_{pj}^{(H)} = \sum_{r=1}^{P} a_{rp} w_{pj} \tag{8.87}$$

- Zotrieď stĺpce matice $Q^{(H)}$ podľa:

$$\left\| S^{(H)}(\vec{c}_1) \right\| \geq \left\| S^{(H)}(\vec{c}_2) \right\| \geq \cdots \geq \left\| S^{(H)}(\vec{c}_p) \right\| \tag{8.88}$$

- Stĺpec s najväčšou hodnotu citlivosti sa zvolí za H-tý stĺpec matice $Q^{(H)}$.
- Spočítaj odchýlku siete $Error^{(H)}$ pre zvolené RBF stredy.

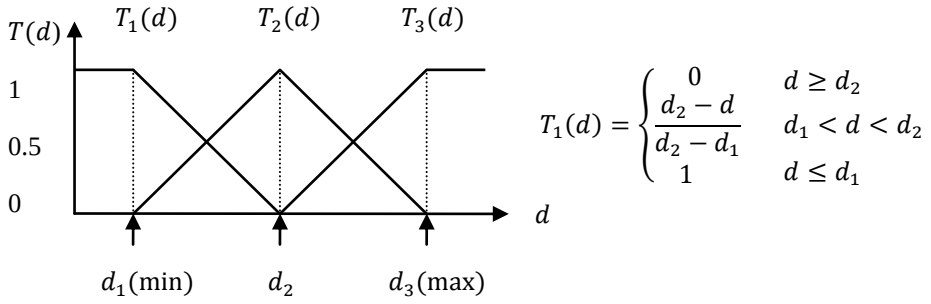
5. Ukončenie algoritmu:
   Ak hodnota rozdielu $Error^{(H)} - Error^{(H-1)}$ je menšia ako vopred preddefinovaná hodnota, tak ukonči algoritmus. V opačnom prípade zvýš hodnotu $H = H + 1$ a pokračuj na krok algoritmu č. 3.

Kritické vektory odpovedajúce prvým H stĺpcom matice $Q^{(H)}$. Práve tieto vektory sa zvolia za stredy RBF jednotiek.

8.9. Kontextové klastrovanie

Kontextové klastrovanie je modifikáciou FCM klastrovacej metódy (4). Kontextové klastrovanie [38] použijeme na rozdelenie vstupných vzorov do klastrov, pričom pri tvorbe týchto klastrov sa berie v úvahu množina preddefinovaných fuzzy množín (nazývaných ako kontexty), ktoré vznikli vo výstupnom priestore.

Nech $T = \{[\vec{x}_p, \vec{d}_p]; 1 \leq p \leq P,\ \vec{x}_p \in R^I, \vec{d}_p \in R^O\}$ je množina P trénovacích vzorov. Najskôr vo výstupnom priestore rozdelíme výstupné hodnoty do klastrov pomocou algoritmu K-means clustering (0) nasledujúcim spôsobom. K-means algoritmom rozdelíme výstupné dáta do $K - 2$ klastrov. Najmenšiu a najväčšiu hodnotu výstupných dát prehlásime za ďalšie dva centroidy, čiže celkovo takto dostaneme K klastrov reprezentovaných centroidami $\vec{c}_k, k = 1, \ldots, K$. Nad týmito centroidami preložíme trojuholníkovú členskú funkciu, tak ako vidieť na obrázku č. 17. Modusy tejto funkcie sú umiestnené v centroidoch $\vec{c}_k$. Členské funkcie označíme $T_1, \ldots, T_K$. Hodnotu členskej funkcie výstupu d_p v j-tom kontexte fuzzy množiny označíme t_{jp}.



Obrázok č. 19: Trojuholníková členská funkcia s polovičným prekrývaním sa po sebe idúcich fuzzy množín. V tomto prípade je počet klastrov $K = 3$ [38].

V ďalšom kroku prevedieme klastrovanie vstupných vzorov pomocou kontextového klastrovania, pričom berieme v úvahu fuzzy množinu kontextov získanú vo výstupnom priestore.

Členskú maticu j-tého kontextu označíme $U(T_j)$ a definujeme predpisom:

$$U(T_j) = \left\{ u_{kp} = [0,1], \sum_{k=1}^{K} u_{kp} = t_{jp}\ \forall p, \quad 0 < \sum_{p=1}^{P} u_{kp} < P\ \forall k \right\} \tag{8.89}$$

K určuje počet klastrov a t_{ji} určuje hodnotu členskej funkcie i-tého vzoru v j-tom kontexte. Chybová funkcia v kontextovom klastrovaní je definovaná predpisom:

$$E_{CFCM} = \sum_{p=1}^{P}\sum_{k=1}^{K} u_{kp}^{m} \left\|\vec{x}_p - \vec{c}_k\right\|^2, \qquad 1 \leq m < \infty, \tag{8.90}$$

m je fuzzyfikačný parameter.

Minimalizáciu chybovej funkcie E_{CFCM} podobne ako v FCM algoritme dosiahneme iteratívnym prepočítavaním hodnôt prvkov členskej matice u_{ki} a pozícií centroidov $\vec{c}_k$. Aktualizáciu hodnôt prvkov členskej matice prevedieme podľa vzťahu:

$$u_{ki} = \frac{t_{jp}}{\sum_{l=1}^{K}\left(\frac{\left\|\vec{x}_p - \vec{c}_k\right\|^2}{\left\|\vec{x}_p - \vec{c}_l\right\|^2}\right)^{\frac{2}{m-1}}}, \tag{8.91}$$

$$k = 1, \ldots, K, p = 1, \ldots P$$

Aktualizáciu pozícií jednotlivých centroidov prevedieme podľa vzťahu:

$$\vec{c}_k = \frac{\sum_{p=1}^{P} u_{kp}^{m}\vec{x}_p}{\sum_{p=1}^{P} u_{kp}^{m}}, \qquad k = 1, \ldots, K \tag{8.92}$$

8.9.1. Redukcia dimenzionality vstupného priestoru

Cieľom redukcie dimenzionality vstupného priestoru je eliminovať čo možno najväčšie množstvo vstupných premenných, ktoré sa predkladajú na vstupy RBF sieti. Zníženie počtu vstupných premenných vedie k jednoduchším architektúram RBF sieti. Zároveň sa z tohto dôvodu skracuje čas potrebný na návrh architektúry a na proces učenia RBF siete.

Eliminácia vstupných premenných sa však tiež musí riadiť istými pravidlami. Po eliminácií vstupných premenných musí platiť, že pomocou premenných, ktoré zostali, bude stále možné riešiť pôvodný problém s rovnakou alebo vyššou požadovanou presnosťou. Eliminovať sa budú hlavne tie premenné, ktoré sú bezvýznamné v zmysle, že nemajú zásadný vplyv v procese návrhu architektúry RBF siete ani na konečné výsledky tejto RBF siete.

Uvažujme dva vstupné vzory $\vec{x}_p, \vec{x}_q$. Označme stĺpce členskej matice, ktoré odpovedajú týmto vzorom ako U_p a U_q. Vzájomnú blízkosť (proximity) týchto vstupných vzorov označíme ako $Prox(U_p, U_q)$, ktorú môžeme vyjadriť vzťahom:

$$Prox(U_p, U_q) = B_{p,q}$$
$$B_{p,q} = \sum_{k=1}^{K} \min(u_{kp},\ u_{kq}),\ \ p, q = 1, \dots, P \tag{8.93}$$

Funkcia vyjadrujúca vzájomnú blízkosť dvoch vzorov je symetrická a vracia jednotku pre rovnaké vstupné vzory a taktiež v prípade, že hodnoty stupňov členstva týchto vzorov do jednotlivých klastrov sú zhodné.

Označme maticu blízkosti v kontexte j ako $U_{(j)}$ a platí $Prox(U_{(j)}) = [B_{p,q}]$.

Naším cieľom je znížiť dimenziu vstupného priestoru, uvažujme preto nejakú podmnožinu vstupných parametrov. Pre vstupné vzory s parametrami z tejto podmnožiny spočítame opäť členskú maticu $U'_{(j)}$ predpisom:

$$u'_{kp} = \frac{t_{jp}}{\sum_{l=1}^{K} \left(\frac{\left\| \vec{x'}_p - \vec{c'}_k \right\|^2}{\left\| \vec{x'}_p - \vec{c'}_l \right\|^2} \right)^{\frac{2}{m-1}}},$$
$$k = 1, \dots, K, p = 1, \dots P \tag{8.94}$$

$\vec{x'}_p$ sú vstupné vzory s vybranými vstupnými parametrami. $\vec{c'}_k$ sú centroidy klastrov pre množinu vstupných vzorov s vybranými vstupnými parametrami, ktoré spočítame podľa vzťahu (6.55).

Na určenie podmnožiny vstupných parametrov je možno použiť napríklad genetický algoritmus. Informácie o genetických algoritmoch možno nájsť v literatúre ([35],[19],[22],[48]).

Pomocou matice $U'_{(j)}$ spočítame maticu blízkosti pre zvolené vstupné parametre podľa vzťahu:

$$Prox(U'_{(j)}) = [B'_{p,q}]$$
$$B'_{p,q} = \sum_{k=1}^{K} \min(u'_{kp}, u'_{kq}), \qquad p, q = 1, \dots, P \tag{8.95}$$

Následne určíme vzdialenosť medzi maticami blízkosti podľa vzťahu:

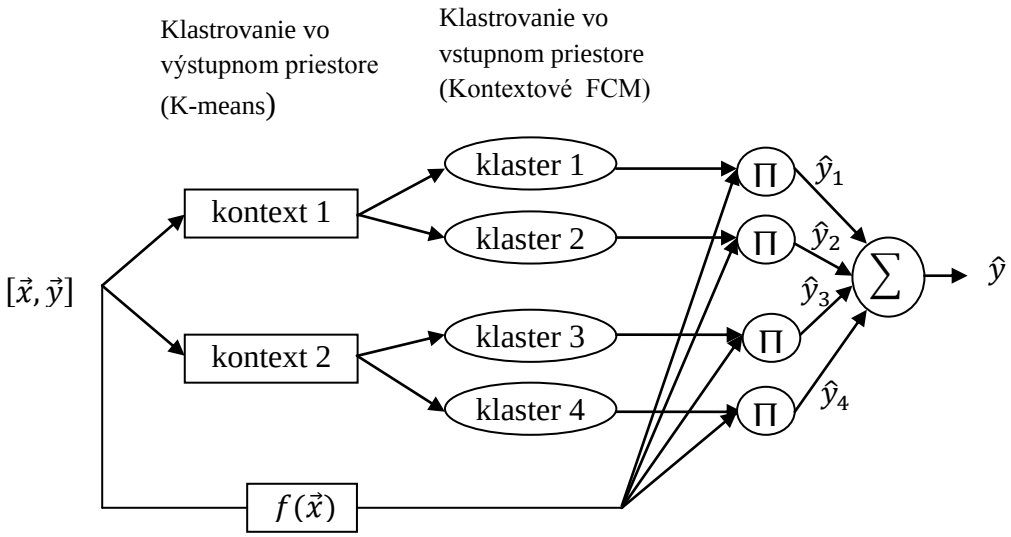
$$D_j = \left\| Prox\left(U_{(j)}\right) - Prox\left(U'_{(j)}\right) \right\| = \left\| B_{p,q} - B'_{p,q} \right\| = \\ = \sum_{p=1}^{P} \sum_{q>p}^{P} \left| B_{p,q} - B'_{p,q} \right| \tag{8.96}$$

Táto vzdialenosť D_j vyjadruje, ako veľmi sa zmenilo rozmiestnenie vstupných vzorov s odobranými parametrami do klastrov oproti pôvodným vstupným vzorom. Naším cieľom je, aby rozdelenie vstupných vzorov s odobranými parametrami do klastrov bolo čo najviac podobné rozdeleniu pôvodných vstupných vzorov, čiže chceme minimalizovať hodnotu D_j.

Vstupné vzory majú dimenziu rovnú I. Chceme vybrať vhodnú podmnožinu I_{opt} parametrov vstupných vzorov, aby vybrané vzory boli menšej dimenzie ako pôvodné vzory. Výber vhodnej podmnožiny I_{opt} vstupných parametrov sa prevedie opäť pomocou genetických algoritmov. Minimalizáciu hodnoty D_j môžeme zapísať:

$$I_{opt} = \arg\min_{\mathrm{I}} D_j \tag{8.97}$$

Topológia RBF siete



Obrázok č. 20: Príklad RBF siete založenej na koncepte kontextového klastrovania. Obsahuje dva klastre pre každý kontext.

Štruktúru takejto RBF siete môžeme zapísať pomocou množiny pravidiel R^n, $n = 1,2, \dots j \times k$. Definujme n-té pravidlo R^n nasledovne:

R^n: ak vstupný vzor $\vec{x}$ náleží do k-tého klastra a j-tého kontextu, tak výstupnú hodnotu $\hat{y}_n$ určíme podľa vzťahu:

$$\hat{y}_n = m_j + a_n\left(\vec{x}_{(j)} - \vec{c}_k\right) \tag{8.98}$$

$\hat{y}_n$ je výstupná hodnota n-tého lokálneho modelu. m_j je modus kontextu j a $\vec{x}_{(j)}$ je vstupný vzor j-tého kontextu.

Ak uvážime všetky tieto pravidla R^n, môžeme vyjadriť celkový výstup siete nasledujúcim predpisom:

$$\begin{aligned}\hat{y}(\vec{x}) = &\sum_{i=1}^{k} u_n(\vec{x})\left[m_1 + a_n^T\left(\vec{x}_{(1)} - \vec{c}_n\right)\right] \\ &+ \sum_{n=k+1}^{2k} u_n(\vec{x})\left[m_2 + a_n^T\left(\vec{x}_{(2)} - \vec{c}_n\right)\right] \\ &+ \sum_{n=(j-1)k+1}^{j\times k} u_n(\vec{x})\left[m_j + a_n^T\left(\vec{x}_{(j)} - \vec{c}_n\right)\right]\end{aligned} \tag{8.99}$$

Chybová funkcia, určujúca výkonnosť siete je definovaná predpisom:

$$\text{RMSE}= \sqrt{\frac{1}{P}\sum_{p=1}^{P}\left(\vec{y}_p - \hat{y}_p\right)^2} \tag{8.100}$$

9. Experimenty s dátami

V tejto kapitole pristúpime k testovaniu predstavených klastrovacích metód K-means algoritmus (0), FCM algoritmus (4), hybridný učiaci proces RBF sietí (7.5), algoritmus EMRAN-RBF (8.3), algoritmus RGAP-RBF (8.4.1). Kohonenové mapy a ich varianty s adaptívnou topológiou typické klastrovanie neprevádzajú. K získaniu rozdelenia vstupných dát do klastrov je naviac nutné použiť nejakú klastrovaciu metódu. Z tohto dôvodu Kohonenové mapy v nasledujúcich testoch vynecháme.

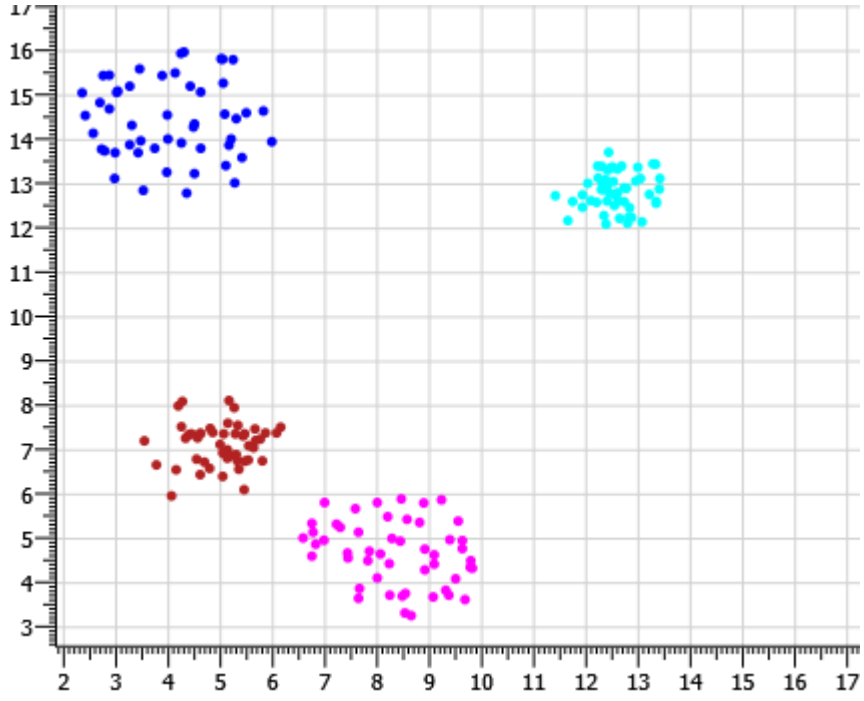
U jednotlivých metód budeme pozorovať správnosť určenia počtu klastrov pre dané vstupné dáta a správnosť rozdelenia týchto vstupných dát do jednotlivých klastrov. Zaujímať nás bude aj čas, ktorý potrebujú tieto metódy k svojmu behu. Práve veľká časová náročnosť môže spôsobiť, že daná metóda je na dátach väčšieho objemu prakticky nepoužiteľná. Algoritmy týkajúce sa RBF sieti sú typu učenia s učiteľom, výsledný počet klastrov je teda už dopredu známy. U týchto algoritmov nás bude zaujímať, koľko RBF jednotiek budú potrebovať k dosiahnutiu požadovanej presnosti RBF siete.

K testovaniu klastrovacích metód použijeme dve rôzne množiny dát. Ako prvú použijeme množinu dát, ktorá obsahuje umelo vygenerované dáta (9.1). Na základe získaných informácií o fungovaní zmienených klastrovacích metód prevedieme klastrovanie na reálnych dátach (9.2), ktoré popisujú vzájomný obchod medzi jednotlivými štátmi sveta.

9.1. Umelo vygenerované dáta

Testovacia množina obsahuje dvojrozmerné vstupné dáta. Vstupné vzory sú teda tvorené dvoma zložkami, ktoré vyjadrujú x-ovú a y-ovú súradnicu v dvojrozmernom priestore, čo nám umožňuje jednoduchú vizualizáciu dosiahnutých výsledkov. Počet vstupných vzorov je 200, pričom rozmiestnenie dát vytvára v priestore 4 disjunktné klastre. Vstupné dáta v dvoch klastroch vznikli pomocou normálneho rozdelenia a v dvoch klastroch pomocou rovnomerného rozdelenia. Každý klaster obsahuje rovnaký počet vstupných vzorov, a to 50.

Rozmiestnenie vstupných dát zachytáva Obrázok č. 21. Od pohľadu je zrejme rozlíšenie jednotlivých klastrov, preto by jednotlivé metódy nemali mať problém identifikovať jednotlivé klastre.



Obrázok č. 21: Umelo vygenerované vstupné dáta, jednotlivé klastre sú farebne odlíšené. Každý klaster obsahuje 50 vstupných vzorov.

Pred samotným testovaním klastrovacích algoritmov je potrebné vykonať určité predspracovanie vstupných dát. Bežne používanou metódou predspracovania dát je normalizácia vstupných dát. Existuje viacero normalizačných metód, my budeme používať min-max normalizáciu na interval $< A, B >$. Min-max normalizácia lineárne transformuje vstupné dáta typicky na interval $< 0, 1 >$. Transformáciu jednotlivých vstupných prevedieme po jednotlivých zložkách testovacích vzorov podľa nasledujúceho vzťahu:

$$x' = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \tag{9.1}$$

x je pôvodná hodnota, x_{max} je maximálna hodnota cez všetky vzory a x_{min} je minimálna hodnota cez všetky vzory. Aby sme mohli vykonať min-max normalizáciu, potrebujeme poznať minimálne a maximálne hodnoty testovacej množiny. Takže v prvom priechode testovacou množinou určíme tieto extrémy a v druhom prechode testovacou množinou vykonáme samotnú min-max normalizáciu.

Proces normalizácie vstupných dát je obzvlášť dôležitý pri klastrovacích úlohách, a to z dôvodu, že tieto klastrovacie metódy počas svojho priebehu počítajú vzdialenosť jednotlivých vstupných vzorov od stredu zhluku. Prevedením normalizácie vstupných dát predídeme vzniku veľkých hodnôt.

Po prevedení min-max normalizácie vstupných dát testovacej množiny môžeme prejsť k samotnému testovaniu vyššie uvedených klastrovacích algoritmov.

K-means algoritmus

K-means algoritmus rozdeľuje vstupné vzory do navzájom disjunktných klastrov. Ako je u K-means algoritmu známe, často sa zastaví v nejakom lokálnom optime, avšak už po niekoľkých iteráciách. Z toho dôvodu tento algoritmus nenájde vždy optimálne riešenie. Vďaka pomerne veľkej rýchlosti tohto algoritmu môžeme tento problém čiastočne odstrániť viacnásobným spustením algoritmu na rovnakých dátach, ale s iným náhodným rozložením centroidov. Na zvolenej testovacej množine spustíme K-means algoritmus jeden tisíckrát. Necháme K-means algoritmus určiť optimálny počet klastrov, pričom počet klastrov sa volí z intervalu 1 až 7.

Celkový čas (s)	Priemerný čas jedného behu (s)	Počet klastrov						
		1	2	3	4	5	6	7
21,3	0,004	0	0	113	648	52	85	102

Tabuľka č. 1: Počet prípadov, v ktorých K-means algoritmus určil jednotlivé počty klastrov za optimálne riešenie.

K-means algoritmus v 65% určil správny počet klastrov, a to 4. Výsledok je pomerne jasný, avšak pre takto jednoduché vstupné dáta sa dajú očakávať lepšie výsledky. Problém nastáva pri iniciálnom rozmiestnení centroidov, ktoré prebieha náhodne. Preto K-means algoritmus v rôznych behoch pri rovnakom počte klastrov na rovnakých vstupných dátach vracia rôzne rozdelenie vstupných dát do klastrov.

FCM algoritmus

FCM algoritmus narozdiel od K-means algoritmu umožňuje, aby vstupný vzor patril do dvoch alebo viacerých klastrov zároveň. U tohto algoritmu sa naviac zameriame na vplyv fuzzyfikačného parametra m na výsledok . Tento parameter určuje ako veľmi sa mení príslušnosť vstupných vzorov ku klastrom vzhľadom k ich vzdialenosti od centroidov jednotlivých zhlukov. Otestujeme FCM algoritmus pre hodnoty $m = \{1.1, 1.5, 2.0\}$. Hodnota ukončovacieho kritéria je zvolená $\varepsilon = 0{,}1$. FCM algoritmus opäť spustíme jeden tisíckrát.

Kvalitu rozdelenia vstupných vzorov z testovacej množiny do klastrov budeme hodnotiť podľa pozmenenej verzie koeficientu členskej funkcie $F'_K(U)$ a podľa Windhamovho proporčného exponentu $W_K(U)$.

$$F'_K(U) = 1 - \frac{K}{K-1}\left(1 - F_K(U)\right) \tag{9.2}$$

$F_K(U)$ je koeficient členskej funkcie zadaný predpisom:

$$F_K(U) = \frac{\sum_{i=1}^{P}\sum_{j=1}^{K} u_{ij}^2}{P} \tag{9.3}$$

$$W_K(U) = -\sum_{i=1}^{P} ln\left[\sum_{k=1}^{\lfloor \mu_i{}^{-1} \rfloor} (-1)^{k+1}\binom{K}{k}(1 - k.\mu_i)^{K-1}\right] \tag{9.4}$$

$$\mu_i = \max_{1 \le k \le K}\{u_{ki}\}$$

Fuzzyfikačný parameter m	Validačné kritérium	Celkový čas (s)	Priemerný čas jedného behu (s)	Počet klastrov						
				1	2	3	4	5	6	7
1.1	$F_K'(U)$	259,1	0,043	0	11	431	493	48	17	0
1.1	$W_K(U)$	274,6	0,046	0	15	387	455	79	29	35
1.5	$F_K'(U)$	272,3	0,045	0	0	173	821	6	0	0
1.5	$W_K(U)$	303,0	0,051	0	0	327	609	5	11	48
2.0	$F_K'(U)$	124,9	0,021	0	33	251	448	159	68	41
2.0	$W_K(U)$	175,4	0,029	0	56	76	401	187	128	152

Tabuľka č. 2: Počet prípadov, v ktorých FCM algoritmus určil jednotlivé počty klastrov za optimum. Hodnoty fuzzyfikačného parametra $m = \{1.1, 1.5, 2.0\}$.

Z výsledkov uvedených v tabuľke vyplýva, že hodnota fuzzyfikačného parametra má nezanedbateľný vplyv na konečné výsledky FCM algoritmu. Z tabuľky vidíme, že pre hodnotu fuzzyfikačného parametra $m = 1{,}5$ bol oboma validačnými kritériami najviackrát $(F_K'(U) - 82\%,\ W_K(U) - 61\%)$ zvolený optimálny počet klastrov rovný 4. Pri hodnotách fuzzyfikačného parametra 1,1 a 2,0 už nie sú výsledné hodnoty také jednoznačné, ale stále je preferovaný počet klastrov 4. Zásadný vplyv fuzzyfikačného prarametra možno pozorovať na časovú náročnosť algoritmu. Pri nižších hodnotách $m \in \{1{,}1;\ 1{,}5\}$ sa doba behu algoritmu výrazne predlžuje, takmer trojnásobne. Z tohto dôvodu je lepšie voliť vyššie hodnoty fuzzyfikačného parametra. Porovnanie validačných kritérií vyznieva lepšie pre koeficient členskej funkcie $F_K'(U)$, algoritmus pri tomto kritériu dosahuje lepšie výsledky, čo sa týka rýchlosti algoritmu aj správnosti výsledného klastrovania.

Porovnajme výsledky K-means algoritmu a najlepšie dosiahnuté výsledky FCM algoritmu ($m = 1{,}5$, validačné kritérium = koeficient členskej funkcie $F_K'(U)$). FCM algoritmus dosahuje presnejšie výsledky, avšak u FCM algoritmu je zjavný nárast časovej zložitosti oproti K-means algoritmu.

Klastrovacia metóda	Celkový čas (s)	Priemerný čas jedného behu (s)	Počet klastrov						
			1	2	3	4	5	6	7
K-means	21,3	0,004	0	0	113	648	52	85	102
FCM	272,3	0,045	0	0	173	821	6	0	0

Tabuľka č. 3: Porovnanie dosiahnutých výsledkov algoritmov K-means a FCM.

EMRAN-RBF algoritmus

Algoritmus EMRAN využíva pri pridávaní a odstraňovaní RBF jednotiek systém posuvných okien. Voľba vhodnej veľkosti posuvného okna je však veľmi závislá na rozdelení vstupných dát. Zároveň je potrebné predpokladať náhodný výber trénovacích vzorov. Ak by totižto boli RBF sieti postupne predkladané vzory z jednotlivých klastrov, musela by byť veľkosť posuvného okna pre odstraňovanie RBF jednotiek veľmi veľká, takmer rovná počtu všetkých predkladaných vzorov. V opačnom prípade by sme postupne mazali RBF jednotky objavené na začiatku algoritmu.

Pred samotným spustením algoritmu je potrebné zadať požadované hodnoty viacerých parametrov a kritérií, ktorých celkový počet je 13. Nastaviť takéto množstvo parametrov je časovo veľmi náročne, pretože neexistuje nejaké univerzálne nastavenie parametrov, aby algoritmus EMRAN fungoval správne. Je potrebné preto využiť všetky dostupné informácie o dátach, ktoré sa majú spracovať. Množstvo a typ rozdelenia vstupných dát pomôže pri voľbe veľkostí posuvných okien. Skrytým pomocníkom je predspracovanie dát, ktoré sme previedli. Konkrétne transformácia dát do intervalu $< 0, 1 >$. Po prevedení tejto transformácie máme jasnú informáciu o minimálnych a maximálnych vzdialenostiach, ktoré môžu nastať medzi predkladanými vzormi a jednotlivými RBF jednotkami. Túto informáciu využijeme pri nastavovaní kritérií pre pridanie, prípadne odstránenie RBF jednotky. Využitie dostupných informácií o vstupných dátach nám síce pomôže približne odhadnúť potrebné hodnoty parametrov, avšak finálne vyladenie hodnôt jednotlivých parametrov získame zrejme jedine pomocou metódy pokusu a omylu.

Po odhadnutí a nastavení hodnôt jednotlivých parametrov sme spustili algoritmus EMRAN jeden tisíckrát na testovacej množine, pričom sme dostali nasledujúce výsledky.

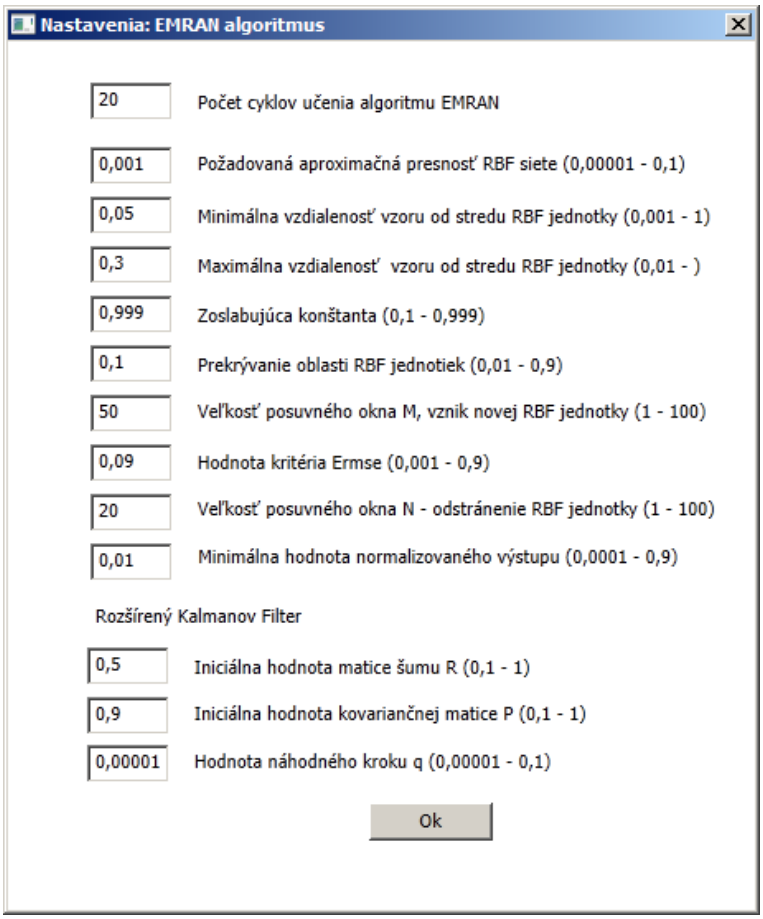
Počet cyklov	Celkový čas (s)	Priemerný čas jedného behu (s)	Počet RBF jednotiek						
			1	2	3	4	5	6	7
1	58,9	0,059	0	0	0	836	149	15	0
10	655,8	0,656	0	0	0	821	177	2	0
20	1072,1	1,072	0	0	0	912	88	0	0

Tabuľka č. 4: Dosiahnuté výsledky na testovacej množine algoritmom EMRAN.

Dosiahnuté výsledky potvrdzujú vhodné nastavenie parametrov. Pre počet cyklov EMRAN algoritmu rovný 10 postačovalo v 91% testoch použiť štyri RBF jednotky k dosiahnutiu požadovanej presnosti RBF siete. Na takto triviálnych vstupných dátach funguje algoritmus EMRAN veľmi dobre, aj keď sú vstupné vzory predložené na vstup RBF sieti iba jedenkrát. Časová náročnosť algoritmu však s narastajúcim počtom cyklov výrazne rastie. Veľkosť posuvných okien pre pridanie, prípadne odstránenie RBF

jednotiek bola pre túto testovaciu množinu relatívne malá, a preto vplyv na výsledný čas je takmer zanedbateľný.

Pre testovaciu množinu umelo vygenerovaných dát boli v algoritme EMRAN použité nasledujúce hodnoty parametrov a kritérií.



Obrázok č. 22: Nastavenia parametrov pre algoritmus EMRAN pre testovaciu množinu umelo vygenerovaných dát.

RGAP-RBF algoritmus

Rekurzívna verzia algoritmu GAP (8.4) umožňuje opakovane predkladať RBF sieti na vstup už v minulosti predložené trénovacie vzory. Oproti algoritmu EMRAN (8.3) prináša zjednodušenie podmienok pre pridanie a odstránenie RBF jednotiek, môžeme teda očakávať rýchlejší priebeh algoritmu. Ďalším kladom oproti algoritmu EMRAN je zníženie počtu parametrov, ktoré je potreba nastaviť pred samotným spustením algoritmu. Týmto znížením počtu parametrov približne o tretinu sa dostávame k potrebe nastaviť deväť parametrov pred spustením algoritmu. Tu opäť ako v algoritme EMRAN platí, že pri nastavovaní požadovaných parametrov je potreba využiť všetky dostupné informácie o vstupných dátach, nevyhneme sa zrejme ani metóde pokusu a omylu.

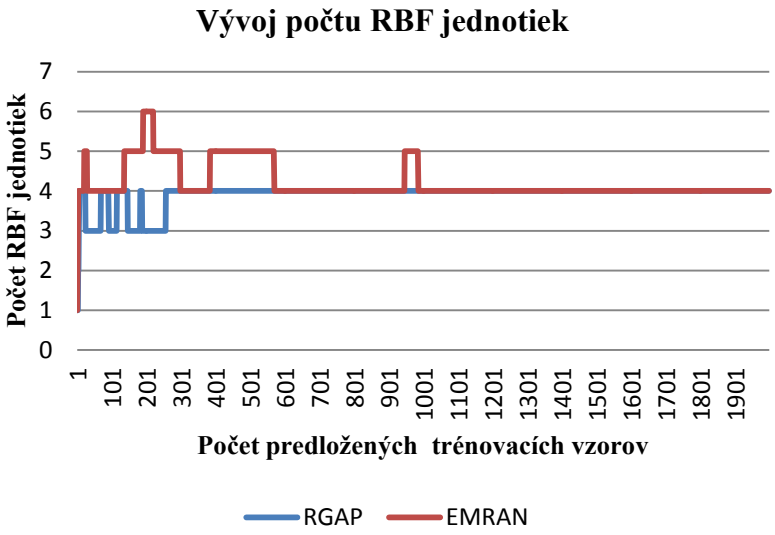
Po odhadnutí a nastavení hodnôt jednotlivých parametrov sme spustili algoritmus RGAP jeden tisíckrát na testovacej množine, pričom sme dostali nasledujúce výsledky.

Počet cyklov	Celkový čas (s)	Priemerný čas jedného behu (s)	Počet RBF jednotiek						
			1	2	3	4	5	6	7
1	42,4	0,042	0	0	170	734	93	3	0
10	473,0	0,473	0	0	59	822	114	5	0
20	953,2	0,953	0	0	47	771	171	11	0

Tabuľka č. 5: Dosiahnuté výsledky na testovacej množine algoritmom RGAP.

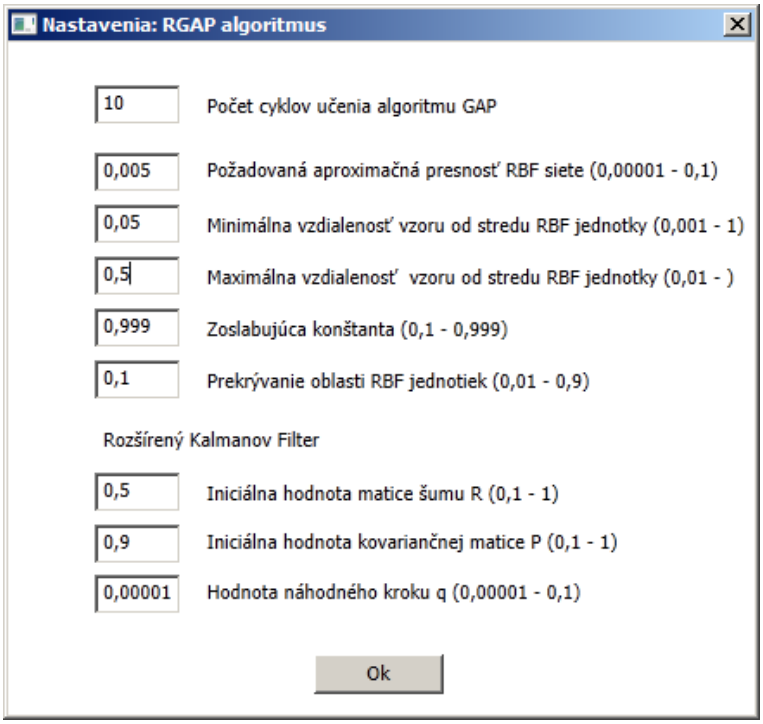
Podobne ako u algoritmu EMRAN je na zvolenej testovacej množine dát dostačujúce predložiť trénovacie vzory iba jeden krát, pričom algoritmus RGAP funguje pre počet cyklov rovný 1 ako klasický GAP (8.4) algoritmus. Napriek získaným výsledkom sa obvykle na vstup RBF sieti predkladajú opakovane trénovacie vzory, čo vedie k dosiahnutiu presnejších RBF sieti.

Nasledujúci graf zobrazuje vývoj počtu RBF jednotiek v závislosti na počte predložených trénovacích vzorov pre algoritmy EMRAN a RGAP. Množina trénovacích vzorov zostala nezmenená, stále sú používané umelo vygenerované vstupné dáta, ktoré obsahujú 200 trénovacích vzorov. Počet cyklov je u oboch algoritmov rovný 10, čiže máme množinu s 2000 trénovacími vzormi. Zachytená je práve situácia, keď po prvom predložení trénovacích vzorov (200 = 1 cyklus) ešte nie je ustálený počet RBF jednotiek. S narastajúcim počtom opakovane predložených trénovacích vzorov sa počet RBF jednotiek postupne ustáli na požadovanom počte 4. Z tohto dôvodu k získaniu presnejších výsledkov je potreba trénovacie vzory predložiť RBF sieti viac ako jeden krát. Z grafu možno vypozorovať, že počet RBF jednotiek sa v algoritme RGAP ustáli oveľa skôr ako v algoritme EMRAN, čo prináša stabilnejšie výsledky RBF siete.



Graf č. 2: Vývoj počtu RBF jednotiek v závislosti na počte predložených trénovacích vzorov pre algoritmy EMRAN a RGAP

Pre testovaciu množinu umelo vygenerovaných dát boli v algoritme RGAP použité nasledujúce hodnoty parametrov a kritérií.



Obrázok č. 23: Nastavenia parametrov pre algoritmus RGAP pre testovaciu množinu umelo vygenerovaných dát.

9.2. Reálne dáta

Množina vstupných dát bude v tomto prípade obsahovať reálne dáta z oblasti svetovej ekonomiky, pričom sa zameriame hlavne na vzájomný obchod medzi jednotlivými štátmi sveta. Na základe objemu uskutočnených obchodných transakcií medzi štátmi budeme skúmať prepojenie ekonomík jednotlivých štátov.

Vstupné dáta popisujúce vzájomný obchod jednotlivých štátov sveta sú dostupné z internetovej stránky Medzinárodného menového fondu [53]. Získané dáta z tohto zdroja sú vo forme dátovej matice. Riadky tejto matice odpovedajú štátom, ktoré sú exportérmi a stĺpce matice odpovedajú štátom, do ktorých sa dováža. Čiže prvok matice, ktorý sa nachádza na i-tom riadku a v j-tom stĺpci, vyjadruje objem exportu z i-tého štátu do štátu j. Celková suma všetkých obchodov je vyjadrená v miliónoch amerických dolárov.

Pri pohľade na zdrojové dáta si možno všimnúť, že u niektorých štátoch nemáme uvedené žiadne informácie o vzájomnom obchode so zvyšnými štátmi. Tieto štáty nemusíme vôbec brať v úvahu. Z dátovej matice teda odstránime prázdne riadky a stĺpce odpovedajúce takýmto štátom. Z pôvodného počtu 198 štátov nám zostane 179 štátov sveta.

Množinu vstupných dát bude vo výsledku tvoriť 179 vektorov rovnako veľkej dimenzie. Aby sme mohli použiť model RBF sieti, potrebujeme k dostupným vstupným dátam priradiť požadované výstupy. K získaniu požadovaných výstupov využijeme internetové stránky svetovej banky [54], kde je dostupné členenie všetkých štátov podľa rozličných kritérií. Zvolíme napríklad rozdelenie štátov podľa ich celkových príjmov. Pomocou RBF sieti budeme u jednotlivých štátov skúmať prípadné závislosti objemu exportu, importu na celkovom bohatstve daného štátu.

Na vstupných dátach prevedieme proces normalizácie. Dáta lineárne transformujeme na interval $< 0, 1 >$ použitím min-max normalizačnej metódy. Takto pripravené dáta môžeme následne predložiť na vstup jednotlivým klastrovacím metódam. Každú metódu spustíme 100 krát a bude sledovať aké výsledky dosiahneme.

Ešte pred samotným testovaním dát sa skúsme zamyslieť aké výsledky by sme mali očakávať po prevedení jednotlivých klastrovacích metód. Chceme rozdeliť štáty sveta do jednotlivých klastrov, pričom poznáme veľkosť objemu vzájomných obchodov týchto štátov. V jednotlivých klastroch by sa teda mali objaviť štáty s podobným objemom zrealizovaných obchodov. Je zrejme, že množstvo týchto zrealizovaných obchodov súvisí s aj veľkosťou jednotlivých štátov, pretože väčšie štáty budú vo väčšom množstve vyvážať aj dovážať. Ďalej možno očakávať regionálne prepojenia, susedné štáty obvykle vzájomne obchodujú vo zvýšenej miere.

K-means a FCM algoritmus

K-means algoritmus po prevedení požadovaného počtu cyklov označil za optimálne rozdelenie vstupných vzorov do 6 klastrov. Odlišné výsledky sme dostali u FCM algoritmu. FCM algoritmus preferuje optimálny počet klastrov 2, pri hodnote fuzzyfikačného parametra rovnej 1,5 . Na prvý pohľad vyzerá, že rozdelenie do 2 klastrov nám veľa o vstupnej množine nenapovie. FCM algoritmus opakovane vytvára menšiu skupinu štátov obsahujúcu 10-12 štátov. Jedná sa o štáty s najväčšími objemami prevedených obchodov. Získavame takto predstavu o štátoch, ktoré majú najväčší podiel na celosvetovom objeme obchodov. Uveďme si hlavných predstaviteľov z tejto skupiny ekonomicky silných štátov:

Čína, Francúzsko, Nemecko, India, Taliansko, Japonsko, Rusko, Singapur, Veľká Británia, USA

Druhú skupinu tvoria zvyšné štáty sveta, o ktorých sa pri takto nastavenom klastrovaní príliš nedozvieme. Po prevedení klastrovania pomocou FCM algoritmu do vopred zvoleného počtu klastrov dostávame nasledujúce výsledky. Skupina už zmienených silných štátov zostáva väčšinou zachovaná celá. Objavujú sa však aj nové skupiny štátov, celkom zaujímavá je nasledujúca skupina.

Argentína, Kolumbia, Venezuela, Chile, Kuvajt, Líbya, Nigéria, Česko, Maďarsko, Slovensko.

Objavené štáty jasne preukazujú regionálne prepojenie obchodných vzťahov. Táto skupina štátov však poukazuje na dôležitý fakt, a to je spájanie podobných klastrov do jedného výsledného klastra. Je preto potrebné hľadať súvislosti aj v skupinách štátov, ktoré na prvý pohľad nedávajú väčší zmysel.

Uveďme výsledky K-means algoritmu, ktorý preferoval rozdelenie štátov do 6 klastrov. Prvé dva klastre sú tvorené skupinou štátov, ktorú identifikoval aj FCM algoritmus. Konkrétne prvý klaster tvorí Čína, Nemecko, USA a druhý klaster tvoria štáty Taliansko, Japonsko, Južná Kórea, Holandsko, Rusko, Veľká Británia.

Tretí a štvrtý klaster majú spoločnú vlastnosť, že zoskupujú štáty z určitej oblasti sveta. Čo je samozrejme logické, že štáty, ktoré sú k sebe geograficky bližšie, uzatvoria vzájomne väčší objem obchodov. Tretí klaster je tvorený štátmi Argentína, Brazília, Čile, Kolumbia a Ekvádor. Štvrtý klaster tvoria štáty India, Malajzia, Saudská Arábia, Singapur, Thajsko, Spojené Arabské Emiráty. Štáty 4-tého klastra sa jednak nachádzajú v rovnakej oblasti sveta, ale všetky tieto štáty môžeme považovať za prístavné štáty. Zrejme ich strategická poloha v lodnej preprave výrazne zvyšuje objem uzatvorených obchodov.

Piaty klaster tvoria štáty Austrália a Brazília. Opäť veľké štáty, ktoré vzájomne obchodujú s veľkým počtom štátov. Šiesty klaster je tvorený zvyšnými štátmi sveta.

Aby sme získali aspoň nejaké informácie o zatiaľ nespomenutých štátoch, zväčšíme rozsah klastrov, z ktorého K-means algoritmus hľadá optimálny počet klastrov. Zvýšme tento rozsah na 15 klastrov. Po opätovnom spustení K-means algoritmu bol najčastejšie označený za optimálny počet klastrov 10. Po rozdelení štátov do desiatich klastrov sme opäť získali klastre obsahujúce už spomínané štáty, prípadne len s minimálnou obmenou jednotlivých štátov. Uveďme však niektoré novo vytvorené klastre, kde sa vyskytuje Česká a Slovenská Republika. V klastri obsahujúcom Českú Republiku sa veľmi často vyskytujú štáty Rakúsko, Dánsko, Fínsko a Belgicko.

Slovenská Republika sa vyskytuje v jednom klastri najčastejšie so štátmi Maďarsko, Portugalsko, Švédsko, Rumunsko, Ukrajina.

RBF siete – algoritmy EMRAN, GAP

Aby sme mohli použiť model RBF sieti, potrebujeme k vstupným dátam priradiť požadované výstupy. K získaniu požadovaných výstupov využijeme internetové stránky svetovej banky [54], kde je dostupné členenie všetkých štátov podľa rozličných kritérií. Zvolíme napríklad rozdelenie štátov podľa ich celkových príjmov. Svetová banka delí štáty do štyroch skupín, a to low income, lower middle income, upper middle income, high income.

Pri použití algoritmov EMRAN (8.3) a GAP (8.4) na množine reálnych dát sme narazili na problém časovej náročnosti spojený s veľkou dimenziou vstupných dát. Oba algoritmy v prípade, že nedôjde k pridaniu novej RBF jednotky do RBF siete aktualizujú iba parametre jednej RBF jednotky. Táto aktualizácia parametrov prebieha pomocou algoritmu EKF (8.5), ktorý počas svojho behu vykonáva viacero maticových operácií (8.3.1). Rozmer používaných matíc (kovariančná, gradientná a matica Kalmanovho úžitku) je rovný súčtu dimenzie vstupných dát = 179, dimenzie výstupných dát = 4 a šírky RBF jednotky = 1. Maticové operácie s maticami o rozmeroch 184 sú už časovo náročné.

Algoritmy EMRAN a GAP boli 50 krát spustené na množine reálnych dát. Vstupné dáta boli RBF sieti predložené iba jeden krát. Priemerný čas jedného behu pre algoritmus EMRAN je 183 s a pre algoritmus GAP 171 s. U oboch algoritmov trvá spracovanie jedného trénovacieho vzoru približne jednu sekundu. Oba algoritmy vyžadujú pred spustením nastaviť hodnoty viacerých parametrov a kritérií, na ktoré sú pritom hodne citlivé. Keďže viacero z týchto parametrov sa obvykle nastavuje metódou pokusu a omylu, tak iba samotné zistenie vhodných hodnôt parametrov je časovo veľmi náročne.

10. Program Klastrovacie Techniky

Súčasťou diplomovej práce je aj program Klastrovacie Techniky. Jedná sa o jednoduchú aplikáciu, pomocou ktorej môžeme testovať niektoré z nasledujúcich uvedených klastrovacích metód:

- K-means algoritmus
- Fuzzy C-means algoritmus
- Kohonenové mapy
- Kohonenové mapy s rastúcou mriežkou
- Model rastúcich neurónových plynov
- Hybridný učiaci proces RBF sieti
- Extended RANEKF
- Extended MRAN
- Rekurzívnu verziu algoritmu GAP

Všetky uvedené metódy som samostatne naprogramoval, a teda svojím spôsobom jedinečné.

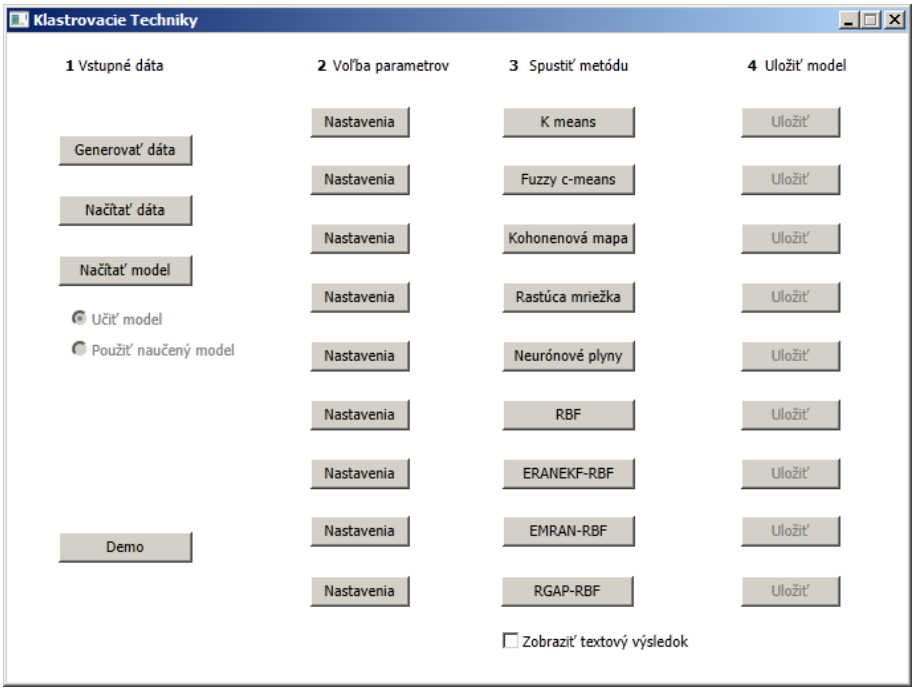
Program Klastrovacie Techniky bol vytvorený v programovacom jazyku C# a funguje na operačnom systéme Windows. K jeho úspešnému spusteniu je potrebné mať nainštalovaný Microsoft.NET Framework 4.

Diskové požiadavky tohto programu sú v dnešnej dobe zanedbateľné, keďže veľkosť tohto programu nepresahuje ani 1MB. Nie sú stanovené minimálne hardwarové požiadavky pre tento program, ale treba upozorniť, že klastrovacie algoritmy patria medzi výpočtovo náročnejšie algoritmy.

Program Klastrovacie Techniky je dostupný na priloženom CD v priečinku Klastrovacie techniky.

10.1.Užívateľské rozhranie

Po spustení aplikácie sa zobrazí hlavné okno.



Obrázok č. 24: Hlavné okno programu Klastrovacie Techniky

Aby sme mohli začať pracovať s jednotlivými metódami, je potreba mať k dispozícii dáta. Dáta môžeme získať dvoma spôsobmi, a to vygenerovať alebo načítať:

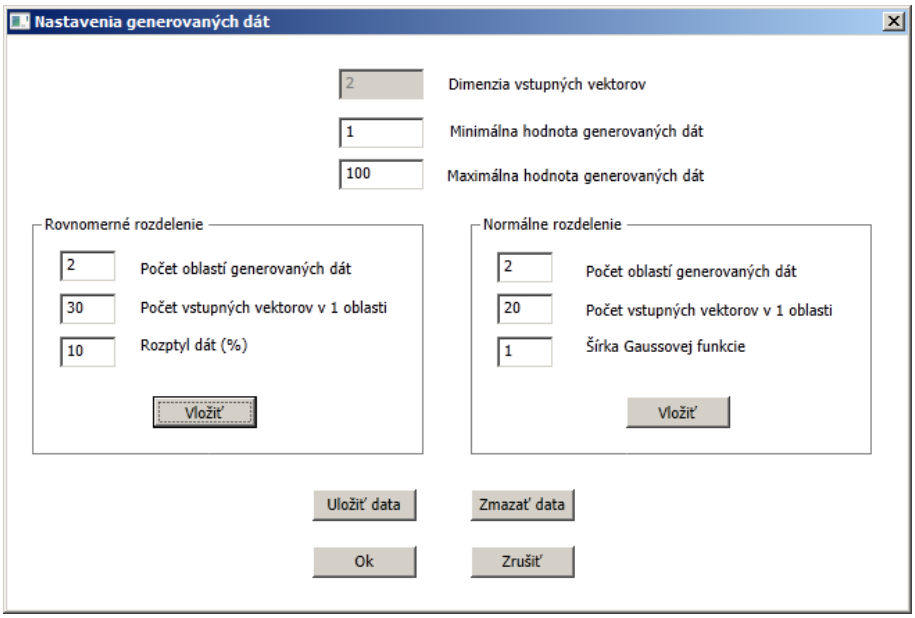
Načítanie dát

Po kliknutí na tlačidlo „Načítaj dáta“ sa zobrazí klasické dialógové okno pre vyhľadanie súboru, s ktorým chceme pracovať. Vstupné dáta sa načítajú z textového súboru, v ktorom je potrebné dodržiavať stanovené formátovanie. Za oddeľovač jednotlivých hodnôt je zvolená dvojica znakov medzera a tabelátor. Každý vstupný vzor sa nachádza na samostatnom riadku textového súboru. Prvá hodnota uvedená na riadku reprezentuje názov vstupného vzoru. Ďalej nasledujú jednotlivé súradnice vstupného vzoru. Ak sú známe aj požadované výstupy, tak na príslušnom riadku textového súboru je ďalej uvedená bodkočiarka – príznak, že nasledujúce hodnoty tvoria súradnice požadovaného výstupu. Prázdne riadky sú ignorované, takže ich výskyt v textovom súbore nespôsobuje žiadny problém. Uveďme si jednoduchý príklad takéhoto textového súboru obsahujúceho 3 vstupné vzory pomenované A, B, C dimenzie 2. Naviac k týmto vstupom máme aj požadované výstupy, nech napríklad vzory A, B patria do jedného klastra a vzor C do patrí do ďalšieho.

A 0.1 0.2 ; 0 1
B 1.1 1.2 ; 0 1
C 2.2 2.3 ; 1 0

Generovať dáta

Druhou možnosťou je pracovať s náhodne generovanými dátami. Po kliknutí na tlačidlo „Generuj dáta“ sa zobrazí nové okno, kde bližšie špecifikujeme požiadavky na generované dáta. Máme na výber, či chceme generovať dáta s rovnomerným alebo normálnym rozdelením. Ďalej nastavíme požadovaný počet klastrov a požadovaný počet vzorov v jednotlivých klastroch. Po kliknutí na tlačidlo „Generuj dáta“ sa vygeneruje množina vstupných vzorov, ktoré spĺňajú špecifikované nastavenia. Pri generovaní vstupných vzorov sa ako názov jednotlivých vstupných vzorov použije poradové číslo vzoru. Vygenerované dáta je možné uložiť kliknutím na tlačidlo „Uložiť dáta“. Ak vygenerované dáta spĺňajú všetky naše požiadavky potvrdíme tlačidlom „Ok“. Pri generovaní vstupných dát sa automaticky generujú aj požadované výstupy. Každému klastru odpovedá jedinečný požadovaný výstup.



Obrázok č. 25: Nastavenie požadovaných hodnôt pri generovaní vstupných dát

Spustiť požadovanú metódu

Po načítaní, prípadne vygenerovaní vstupných dát sa sprístupnia tlačidla umožňujúce spustenie jednotlivých klastrovacích metód. Zároveň sa sprístupnia tlačidla pre nastavenia požadovaných parametrov jednotlivých metód. Kliknutím na tlačidlo „Nastavenia“ sa zobrazí okno, kde je možné zadať požadované hodnoty parametrov príslušnej metódy.

Po získaní vstupných dát a nastavení požadovaných hodnôt spustíme požadovanú metódu kliknutím na tlačidlo, ktorého text odpovedá požadovanej metóde. Po dokončení algoritmu sa zobrazia získané výsledky.

Podľa dimenzie vstupných vzorov sa určuje forma výpisu výsledkov. Ak pracujeme so vstupnými vzormi dimenzie dva môžeme prvú zložku vstupného vzoru považovať za x-ovú súradnicu a druhú zložku za y-ovú súradnicu. Nie je teda problém zobraziť rozdelenie vstupných dát do klastrov aj graficky. Popri grafickom zobrazení je možné získané výsledky zobraziť aj v textovej podobe. Pri textovom výpise sa využijú názvy jednotlivých vstupných vektorov. V prípade vstupných vzorov dimenzie rôznej od dva sa grafické zobrazenie výsledkov neprevádza. Zobrazí sa len textový popis výsledkov.

Obrázok č. 26: Textové zobrazenie rozdelenia štátov sveta do klastrov pomocou FCM algoritmu.

Uložiť vyvinutý model

Po prevedení požadovanej metódy sa sprístupni tlačidlo „Uložiť“, kliknutím na toto tlačidlo sa uložia všetky parametre novovzniknutého modelu. Kedykoľvek sa tak môžeme vrátiť k tomuto modelu a predkladať nové vstupné dáta. Načítanie uloženého modelu prevedieme kliknutím na tlačidlo „Načítať model“. Po získaní modelu

ľubovoľnej metódy sa sprístupní pod tlačidlom „Načítať model“ tzv. radio-button, pomocou ktorého si zvolíme, či chceme ďalej pracovať s naučeným modelom alebo chceme vyvinúť nový model.

10.2. Implementácia programu Klastrovacie Techniky

Grafické rozhranie tohto programu bolo vytvorené pomocou nového grafického framework-u Windows Presentation Foundation. Jedná sa o alternatívny spôsob písania Windows aplikácií k známym Windows Forms. Pri grafickom zobrazovaní výsledkov sa využíva knižnica Microsoft.Research.DynamicDataDisplay. Program ako celok je vytvorený v programovacom jazyku C#.

Samotný program je rozdelený do viacerých tried.

Trieda K_Means.cs implementuje metódy potrebné k prevedeniu K-means klastrovania. Ide hlavne o metódy, ktoré majú na starosti aktualizáciu pozícií stredov jednotlivých zhlukov a ich vzájomné porovnávanie.

Trieda FCM.cs implementuje metódy potrebné k prevedeniu klastrovania pomocou FCM algoritmu. Najdôležitejšie metódy zabezpečujú aktualizáciu prvkov členskej matice a aktualizáciu pozícií stredov jednotlivých klastrov.

Trieda SOM.cs implementuje základné metódy umožňujúce pracovať s neurónmi v topologickej mriežke. Túto triedu dedia triedy implementujúce modely Kohonenových máp, čiže KohonenovaMapa.cs, KM_RastucaMriezka.cs a KM_NeuronovyPlyn.cs

Trieda RBF.cs implementuje metódy a štruktúry umožňujúce previesť hybridný učiaci proces.

Trieda RBFAdaptTopologia implementuje základné metódy potrebná pri algoritmoch umožňujúcich aktualizovať topológiu RBF siete. Túto triedu dedia všetky triedy implementujúce algoritmy pre tento typ RBF sieti RANEKF.cs, EMRAN.cs, GAP.cs.

Trieda MainWindow.xaml.cs zabezpečuje obsluhu hlavného okna programu. Jedná sa o centrálnu triedu, ktorá inicializuje vznik všetkých uvedených tried. Informácie o iniciálnych hodnotách a nastaveniach jednotlivých položiek hlavného okna je možné nájsť v MainWindow.xaml.

Trieda vstupneHodnoty.cs je abstraktná trieda, ktorá udržiava aktuálne hodnoty všetkých parametrov.

11. Záver

V tejto diplomovej práci som zrekapituloval viacero metód, ktoré sa využívajú ku klastrovaniu vstupných dát. Ako prvé som predstavil algoritmy typu učenia bez učiteľa. K použitiu týchto algoritmov nie sú potrebné žiadne znalosti riešenej úlohy. Algoritmus K-means (3) vyniká svojou rýchlosťou, presnosť výsledkov je však závislá na iniciálnom rozmiestnení centroidov. Viacnásobné spustenie algoritmu tento problém eliminuje. Algoritmus FCM (4) je oproti K-means algoritmu pomalší, veľký vplyv na výslednú presnosť a dobu klastrovania má fuzzyfikačný parameter. Pri hodnotách fuzzyfikačného parametra blízko jednotky sa dosahuje vyššia presnosť klastrovania avšak za cenu zvýšených časových nárokov. Je preto nutné hľadať kompromis na oboch stranách, ako vhodná hodnota fuzzyfikačného parametra sa javí hodnota okolo 1,5.

Ďalej som predstavil tri modely Kohonenových máp. Základný model Kohonenových máp (5) musí vykonať veľké množstvo iterácií (obvykle desať tisíce iterácií) kým sa dosiahne rozprestretie a vyrovnanie topologickej mriežky vo vstupnom priestore. Použitie tohto modelu je výhodné v situáciách, keď je množina všetkých pozorovaní vopred známa a v čase sa už táto množina nemení. V takomto prípade sa nechá Kohonenová mapa dôkladne naučiť na trénovacích vzorov. Keďže sa tento proces učenia vykoná iba jeden krát (prípadne len zriedkavo), nie je časová náročnosť tak veľkou prekážkou. Po naučení už funguje Kohonenová mapa relatívne rýchlo. Modely Kohonenových máp s adaptívnou topológiou sú v porovnaní so základným modelom oveľa rýchlejšie. Kohonenové mapy s rastúcou mriežkou (6.1) pridávajú do topologickej mriežky v jednom kroku viacero neurónov súčasne. Zbytočne tak vznikajú nevyužité neuróny. Tento problém odstraňujú Neurónové plyny (6.2), ktoré pridávajú neuróny v prípade potreby iba po jednom a zároveň umožňujú odstránenie neurónov z topologickej mriežky.

Základný model RBF sieti (7) má vopred pevne zvolenú architektúru. K dosiahnutiu požadovanej presnosti sú preto potrebné bližšie znalosti riešenej úlohy. Predstavil som dva algoritmy pre základný model RBF sieti. Prvý algoritmus náhodnej voľby RBF jednotiek (7.4.1) k dosiahnutiu požadovanej presnosti generuje množstvo redundantných RBF jednotiek. Hybridný učiaci proces (7.5) využíva k rozmiestneniu RBF jednotiek Kohonenové mapy. RBF jednotky sú tak umiestňované do oblasti vstupného priestoru, kde je predpoklad zvýšeného výskytu vstupných vzorov. Výskyt redundantných RBF jednotiek sa však stále nedá vylúčiť.

V prípade, že nedisponujeme žiadnymi informáciami o riešenom probléme, je vhodné využiť RBF siete s adaptívnou topológiou (8). Výsledná topológia RBF siete sa postupne vyvíja počas priebehu učiaceho algoritmu. Algoritmy RAN (8.1) a RANEKF (8.1) umožňujú iba vznik nových RBF jednotiek, nevýhodou oboch algoritmov je

opakovaná aktualizácia parametrov všetkých RBF jednotiek, čo vedie k zvýšením časovým nárokom. Algoritmus MRAN (8.2) na rozdiel od algoritmov RAN, RANEKF umožňuje aj odstránenie RBF jednotiek. Výsledná RBF sieť by tak mala obsahovať nižší počet RBF jednotiek. Algoritmus MRAN rovnako ako algoritmy RAN, RANEKF opakovane aktualizuje parametre všetkých RBF jednotiek. Vo fáze odstraňovania RBF jednotiek sa testujú všetky RBF jednotky na prípadne odstránenie, čo naviac vedie k zvýšeným časovým nárokom algoritmu. Algoritmus EMRAN (8.3) prináša zrýchlenie algoritmu MRAN. V prípade, že nedôjde k pridaniu novej RBF jednotky do RBF siete, aktualizujú sa pomocou algoritmu EKF (8.5) iba parametre najbližšej (v Euklidovej vzdialenosti) RBF jednotky. Algoritmus GAP (8.4) prináša ďalšie zrýchlenie oproti algoritmu EMRAN. Vo fáze odstraňovania RBF jednotiek sa testuje na prípadne odstránenie už iba jedna RBF jednotka, a to najbližšia (v Euklidovej vzdialenosti) RBF jednotka k poslednému predloženému vstupnému vzoru. Problém s tzv. „pod trénovaním“ RBF siete môžeme vyriešiť opakovaným predkladaním už v minulosti predložených trénovacích vzorov. Nárast počtu predkladaných vzorov má však vplyv na nárast časovej zložitosti algoritmu.

U všetkých spomenutých algoritmov, ktoré sa týkajú RBF sieti s adaptívnou topológiou natrafíme na problém spojený s iniciálnym nastavením hodnôt parametrov. Celkovo je potreba zadať hodnoty pre 9 až 13 parametrov v závislosti na konkrétnom algoritme. Naviac experimenty na reálnych dátach preukázali, že tieto metódy majú problém pracovať s vstupnými dátami veľkej dimenzie.

12. Použitá literatúra

[1] Antonini M., Barlaud M., Mathieu P.: In Signal Processing V. theories and Applications. Proc EUSIPCO-90, Fifth European Signal Processing Conference, ed. by L. Torres, E. Masgrau, M. A. Lagunas (Elsevier, Amsterdam, Netherlands 1990), p. II-1091.

[2] Backer E., Jain A.K.: A clustering performance measure based on fuzzy set decomposition, IEEE Trans. Patten Anal. Mach. Intell. 3 (1) (1981) 66–74.

[3] Bezdek J. C., Ehrlich R., Full W.: FCM: The fuzzy c-means clustering algorithm, in: Computers & Geosciences Vol.10, No. 2-3, pp. 191-203, 1984.

[4] Bezdek J. C.: Cluster validity with fuzzy sets, J. Cybernet. 3 (1974) 58–73.

[5] Bezdek J. C.: Numerical taxonomy with fuzzy sets, J. Math. Biol. 1 (1974) 57–71.

[6] Bezdek J. C.: Pattern Recognition with Fuzzy Objective Function Algorithms, Plenum Press, New York, 1981.

[7] Broomhead D. S., Lowe D.: Multivariable functional interpolation and adaptive networks. Complex Systems, 2:321-355, 1988.

[8] Cover T. M.: Geometrical and statistical properties of systems of linear inequalities with application in pattern recognition. IEEE Transaction on Electronic Computers, EC-14:326-334, 1965.

[9] Fritzke B.: A Growing Neural Gas Network Learns Topologies, Advances in Neural Information Processing Systems 7, pp. 625-632, MIT PRESS, 1995.

[10] Fritzke B.: Growing Grid - a self-organizing network with constant neighborhood range and adaptation strength, Neural Processing Letters, Vol. 2, pp. 9-13, 1995.

[11] Haykin S.: Neural networks: a comprehensive foundation, Prentice Hall, 1999.

[12] Hong X. and Billings S.: Givens rotation based fast backward elimination algorithm for RBF neural network pruning, *Proc. Inst. Elect. Eng. Control Theory Appl.* 144 (5) (1997), pp. 381–384.

[13] Hong X., Harris C., Brown M. and Chen S.: Backward elimination methods for associative memory network pruning, *Int. J. Hybrid Intell. Syst.* 1 (1) (2004), pp. 90–99.

[14] Huang G. B., Saratchandran P., Sundararajan N.: A Generalized Growing and Pruning RBF (GGAP-RBF) Neural Network for Function Approximation, in: IEEE Transactions on Neural Networks, Vol. 16, No. 1, (2005) pp. 57-67.

[15] Huang G. B., Saratchandran P., Sundararajan N.: A Recursive Growing and Pruning RBF (GAP-RBF) Algorithm for Function Approximations, Control and Automation ICCA'03. Proceedings. 4th International Conference on, pp. 491 – 495, 2003.

[16] Huang S. H.: Dimensionality reduction in automatic knowledge acquisition: A simple greedy search approach, in: IEEE Trans. Knowl. Data Eng., Vol. 15, No. 6, (2003) pp. 1364-1373.
[17] Chen S., Cowan C. F., Grant P. M.: Orthogonal Least Squares Learning Algorithm for Radial Basis Function Network, IEEE Transactions on Neural Networks, vol. 2, no. 2, March 1991.
[18] Jakubowitz O. G.: In Proc. IJCNN'89, Int. Joint Conf. on Neural Networks (IEEE Service Center, Piscataway, NJ, 1989) p. II-23.
[19] Jiao J., Zhang Y., and Wang Y.: "A heuristic genetic algorithm for product portfolio planning," *Comput. Operat. Res.*, vol. 34, pp. 1777–1799, 2007.
[20] Kadirkamanathan V., Niranjan M.: A Function Estimation Approach to Sequential Learning with Neural Networks, Neural Computation, Vol.5, pp. 954-975, 1993.
[21] Kallio K., Haltsonen S., Paajanen E., Rosqvist T., Katila T., Karp P., Malmberg P., Piirilä P., Sovijärvi A. R. A.: In Artificial Neural Networks, ed. By T. Kohonen, K. Mäkisara, O. Simula, J. Kangas (North-Holland, Amsterdam, Netherland 1991) p. I-803.
[22] Kim N. J., Kehtarnavaz N., Yeary M. B. and Thornton S.: "DSP-based hierarchical neural network modulation signal classification," *IEEE Trans. Neural Netw.*, vol. 14, no. 5, pp. 1065–1071, Sep. 2003.
[23] Koh J., Suk M., Bhandarkar S. M.: In Proc. ICNN'93, Int. Conf. on Neural Networks (IEEE Service Center, Piscataway, NJ, 1993) p. III-1270.
[24] Kohonen T.: Self-Organizing Maps, 3rd Edition, Springer-Verlag, 2001.
[25] Kohonen T.: Self-Organizing Maps. In Proc. of IEEE volume 78. September 1990.
[26] Li Yan, Sundararajan n., Saratchandran P.: Analysis of Minimal Radial Basis Function Network Algorithm for Real-Time Identification of Nonlinear Dynamic Systems, IEE Proc.,Vol.147, pp. 476-484, 2000.
[27] Liu X., Cheng G., Wu J.: In Proc. ICNN'94, Int. Conf. on Neural Networks (IEEE Service Center, Piscataway, NJ, 1994) p. 649.
[28] Lu Y. W., Sundararajan N., Saratchandran P.: A sequential Learning Scheme for function Approximation and Using Minimal Radial Basis Neural Networks, Neural Computation, Vol.9, pp. 1-18, 1997.
[29] Luttrell S. P.: Pattern recognition Letters 10, 1 (1989).
[30] Mao K. Z., Huang G.-B.: Neuron selection for RBF neural network classifier based on data structure preserving criterion, in: IEEE Trans. Neural Netw., Vol. 16, No. 6, (2005) pp. 1531-1540.
[31] Martinetz T., Schulten K.: In Proc. ICNN'93, Int. Conf. on Neural Networks (IEEE Service Center, Piscataway, NJ, 1993) p. II-820.
[32] Moody J. and Darken C.: Learning with localized receptive fields. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, Proceedings of

the Connectionist Models Summer School. San Mateo: Morgan Kaufmann, 1989, 1990.
[33] Moody J. and Darken C.: Fast adaptive k-means clustering: some empirical results. In Proc. IJCNN, San Diego '90, volume 2, pages 233-238, 1990.
[34] Morabito M., Macerata A., Taddei A., Marchesi C.: In Proc. Computers in Cardiology (IEEE Comput. Soc. Press, Los Alamitos, CA 1991) p. 181.
[35] Oh S. K., Pedrycz W., and Park H. S.: "Genetically optimized fuzzy polynomial neural networks," *IEEE Trans. Fuzzy Syst.*, vol. 14, no. 1, pp. 125–144, Feb. 2006.
[36] Orr M. J. L. and Hallam J.: Int. J. Neural Syst. 10 5 (2000), pp. 397–415.
[37] Orr M. J. L.: Introduction to radial basis function networks, 1996.
[38] Park H.-S., Pedrycz W., Oh S.-K.: Granular Neural Networks and Their Development Through Context-Based Clustering and Adjustable Dimensionality of Receptive Fields, in: IEEE Transactions on Neural Networks, (2009) 13 p., DOI: 10.1109/TNN.2009.2027319.
[39] Platt, J.C.: A resource Allocating Network for Function Interpolation, Neural Computation, Vol3, pp. 213-225, 1991.
[40] Pedrycz W.: Fuzzy clustering with a knowledge-based guidance, in: Pattern Recognition Letters, Vol. 25, (2004) pp. 469-480.
[41] Powell M. J. D.: Radial basis function for multivariable interpolation: A review. In J. C. Mason and M. G. Cox, editors, Algorithms for approximation, pages 143-167. Oxford Science Publications, 1987.
[42] Rojas I., Pomares H., Bernier J. L., Ortega J., Pino B., Pelayo F. J. and Prieto A.: "Time series analysis using normalized PG-RBF network with regression weights," *Neurocomput.*, vol. 42, pp. 267–285, 2002.
[43] Rojas R.: Neural Networks: A Systematic Introduction, Springer-Verlag, 1996.
[44] Salmerón M., Ortega J., Puntonet C. G. and Prieto A.: "Improved RAN sequential prediction using orthogonal techniques," *Neurocomput.*, vol. 41, pp. 153–172, 2001.
[45] Shi D., Yeung D. S., Gao J.: Sensitivity analysis applied to the construction of radial basis function networks, in: Neural Networks, Vol. 18, (2005) pp. 951-957.
[46] Šíma J., Neruda R.: Teoretické otázky neuronových sítí, MATFYZPRESS, Praha, 1996.
[47] Trauwaert E.: On the meaning of Dunn's partition coefficient for fuzzy clusters, Fuzzy Sets and Systems 25 (1988) 217–242.
[48] Verma B. and Zhang P.: "A novel neural-genetic algorithm to find the most significant combination of features in digital mammograms," *Appl. Soft Comput.*, vol. 7, pp. 612–625, 2007.
[49] Wang W., Zhang Y.: On fuzzy cluster validity indices, in Fuzzy Sets and Systems 158 (2007) 2095-2117.

[50] Windham M. P.: Cluster validity for fuzzy clustering algorithms. Fuzzy Sets and Systems 5 (1981) 177–185.
[51] Yeung D. S., Ng W. W. Y., Wang D., Tsang E. C. C., Wang X. Z.: Localized generalization error model and its application to architecture selection for radial basis function neural network, in: IEEE Transactions on Neural Networks, Vol. 18, No. 5, (2007) pp. 1294-1305.
[52] Sundararajan N., Saratchandran P., Lu Ying Wei: Radial Basis Function Neural Networks with Sequential Learning (MRAN and Its Applications), World Scientific Publishing Co. Pte. Ltd. 1999.

[53] International Monetary Fund: http://www.imf.org/external/index.htm
[54] World bank: http://www.worldbank.org
[55] http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/AppletFCM.html
[56] Wikipedia – Determining the number of cluster in a data set: http://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set#The_Elbow_Method

Dodatok

Obsah priloženého CD

- súbor Diplomová práca.pdf – text diplomovej práce vo formáte pdf

- priečinok „Klastrovacie Techniky - Program“ obsahuje:
  - spustiteľnú verziu programu Klastrovacie Techniky spolu s potrebnými DLL knižnicami
  - Microsoft .Net Framework 4
  - vstupné dáta:
    - umelo vygenerované dáta
    - reálne dáta popisujúce medzinárodný obchod jednotlivých štátov sveta

- priečinok „Klastrovacie Techniky - Zdrojové súbory“ obsahuje zdrojový kód programu Klastrovacie Techniky